Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Desenvolvimento de jogo eletrônico para uso como instrumento para pesquisas na área de comportamento.

Igor Rafael de Sousa
Vinicius Tafuri Froes Carvalho

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof.ª Dr.ª Carla Denise Castanho

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof.ª Dr.ª Coordenadora Maristela Terto de Holanda

Banca examinadora composta por:

Prof.ª Dr.ª Carla Denise Castanho (Orientador) — CIC/UnB
Prof. Dr. Rodrigo Bonifácio — CIC/UnB
Prof. Ms. Mauricio Miranda Sarmet — PST/UnB
Prof. Ms. Tiago Barros Pontes e Silva — DIN/UnB

CIP — Catalogação Internacional na Publicação

de Sousa, Igor Rafael.

Desenvolvimento de jogo eletrônico para uso como instrumento para pesquisas na área de comportamento. / Igor Rafael de Sousa, Vinicius Tafuri Froes Carvalho. Brasília : UnB, 2013.
163 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. Jogos, 2. Piscologia Social, 3. Violência

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Desenvolvimento de jogo eletrônico para uso como instrumento para pesquisas na área de comportamento.

Igor Rafael de Sousa
Vinicius Tafuri Froes Carvalho

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Carla Denise Castanho (Orientador)
CIC/UnB

Prof. Dr. Rodrigo Bonifácio
CIC/UnB

Prof. Ms. Mauricio Miranda Sarmet
PST/UnB

Prof. Ms. Tiago Barros Pontes e Silva
DIN/UnB

Prof.ª Dr.ª Coordenadora Maristela Terto de Holanda
Coordenador do Bacharelado em Ciência da Computação

Brasília, 25 de julho de 2013

Agradecimentos

Em primeiro lugar, agradecemos à nossa orientadora, professora Carla Denise Castanho, que tornou possível a execução deste trabalho. Agradecemos, também, ao professor Mauricio Miranda Sarmet que esteve sempre presente nas discussões e disposto a ajudar nas diversas etapas do trabalho, em especial, na parte de teste. Somos gratos aos professores Rodrigo Bonifácio e Tiago Barros Pontes e Silva por terem aceito o convite para participarem da banca de avaliação deste trabalho. Não podemos deixar de mencionar a grande ajuda dos integrantes do grupo de de pesquisa Pró-Games, cujas pesquisas propiciaram a oportunidade de desenvolvimento deste trabalho e cujas ideias e sugestões acerca do jogo foram fundamentais. Agradecemos, também, a Caue Pascal Kopczinski e Herman Militão Ferreira de Azevedo, que se voluntariaram para fazer o *level design* e música do jogo, respectivamente, tornando esse trabalho possível e elevando a qualidade do produto final. Não podemos deixar de mencionar, também, a empresa Fira Soft que cedeu o espaço e algumas ferramentas para o desenvolvimento do trabalho.

Eu, Vinicius Tafuri Fróes de Carvalho, agradeço aos colegas Victor Henrique Taira, Gracielle Olivier, Wesley Vágner Silva e Matheus Parreiras que sempre me apoiaram e ajudaram durante o curso. Agradeço, também, aos amigos Leandro Pereira e Angelo Rossi por sempre mostrar interesse no trabalho, me motivando a continuar o desenvolvimento desse. Agradeço, também, meu pai, Fernando José Fróes de Carvalho, que ajudou muito na escrita da monografia e minha mãe, Maria Izabel Raso Tafuri, que me apoiou e foi muito compreensiva. Agradeço, também, minha avó, Maria da Conceição Raso Tafuri, e meu falecido avô, Washington Luiz Raso Tafuri por todo amor e sabedoria que me passaram ao longo dos anos, que foram os fatores mais importantes para eu chegar até aqui.

Eu, Igor Rafael de Sousa, agradeço primeiramente a minha mãe, Elza Paula de Sousa, e meu pai, Fausto Onofre de Sousa, que deram toda a base necessária e contínuo apoio para eu chegar até aqui. Agradeço também dos meus colegas que contribuíram para que eu aproveitasse bem meus anos de graduação: André Segóvia, Bruno Lourenço, Caio Castro, Davi Diniz, Eduardo Freire, Felipe Lessa, Herman Ferreira, Jéssica França, Marcelo Asari, Matheus Rocha Pitta e Raul Santana. Essa experiência foi inspirada por professores que, de diferentes maneiras, inspiraram em mim o amor pela computação e desenvolvimento de jogos: Alba Cristina de Melo, Carla Denise Castanho, Guilherme Albuquerque Pinto, José Carlos Loureiro Ralha, Lineu da Costa Araújo, Marcus Vinícius Lamar e Tiago Barros Pontes e Silva. Um agradecimento especial aos amigos que têm contribuições em diversas áreas da minha experiência acadêmica e fora dela: Felipe Modesto, Leonardo Guilherme e Luiggi Reffatti. Agradeço também aos meus inúmeros amigos que me estavam comigo nas horas de lazer e me perdoavam quando eu as transformava em trabalho.

Resumo

Com a popularização de jogos eletrônicos surgiu um crescente interesse pelos seus efeitos sobre os jogadores, particularmente com relação a comportamentos violentos. Para análise desses efeitos existem diversas pesquisas que comparam jogos distintos para estudar sua influência. Nessas pesquisas os jogos utilizados não atendem às demandas dos pesquisadores por conterem diversas influências indesejadas.

Este trabalho objetiva o desenvolvimento de um jogo que possua o mínimo possível de influências indesejadas entre três diferentes modos. Esses modos foram escolhidos para que pudessem substituir os jogos violentos, neutros e pró-sociais atualmente utilizados em pesquisas. Além dessa diferença básica entre os modos, também é possível fazer diversas pequenas customizações no jogo para estudar como as influências afetam os resultados das pesquisas. O trabalho também objetiva o estudo de ferramentas para o controle da dificuldade em jogos. Para alcançar estes objetivos são desenvolvidas ferramentas dentro da engine Unity.

Para auxiliar pesquisas futuras, muitos dados são coletados durante o jogo. Esses dados também são utilizados como alimentação de uma ferramenta de dificuldade dinâmica que altera o comportamento do jogo e do cenário de acordo com a análise dos dados.

Ao final do desenvolvimento foi feito um teste online que mostrou que o jogo cumpre com os propósitos da pesquisa.

Palavras-chave: Jogos, Piscologia Social, Violência

Abstract

As the popularity of digital games grows the interest in their effects on players also grows, specialy regarding violent behaviours. Numerous studies compare different games in order to analyse how they influence players. The games used in these studies do not fulfill all the research needs, for they contain many undesired influences.

This work presents the development of a game with the minimum amount of undesired influences between three different modes. These modes were picked to replace the violent, neutral and pro-social games currently used in studies.

Besides this basic difference between modes, it's also possible to set various influences in order to study how they affect the results of studies. Another goal of this work is the study of tools to control difficulty in games, this is achieved through the development of tools within the Unity game engine.

To assist future studies, data is colected during gameplay. This data is also used as input for a dynamic difficulty tool that changes game and scenario behaviour according to the analysis of the data.

After development an online test was carried out and showed that the game fulfills the study goals.

Keywords: Games, Social Psychology, Violence

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Jogos e brincadeiras sempre fizeram parte do desenvolvimento dos jovens. Desde a popularização de computadores e outros meios capazes de prover entretenimento digitais, os jogos eletrônicos estão se tornando uma parte cada vez mais importantes de nossa sociedade. Essa importância pode não ser óbvia para alguns, mas é facilmente comprovada pelo sucesso econômico atingido por alguns títulos assim como diversas decisões governamentais que reconhecem os jogos eletrônicos como cultura.

Além disso, a indústria dos games amadureceu, de acordo com a *Entertainement Software Association* (ESA) em seu ultimo relatório de pesquisa publicado em 2010 [33], a idade média de jogadores nos EUA é de 36 anos e 78% dos moradores americanos têm videogames em suas casas. Entretanto, o fato mais importante a se ressaltar é que 42% dos entrevistados jogam jogos eletrônicos, um aumento de mais de 100% se comparada com a última pesquisa feita em 2002, onde este número era somente 20%.

Sendo os jogos tão presentes na sociedade atual, muitos pesquisadores investigam seu impacto na vida das pessoas. Para mensurar este impacto são realizadas pesquisas de diversas naturezas que buscam entender possíveis efeitos dos jogos no comportamento do ser humano. A Psicologia vem realizando diversos estudos nesta área [13], em particular o grupo Pró-Games da Universidade de Brasília [5].

A seguir é feita uma breve contextualização teórica acerca dos principais assuntos discutidos, a Psicologia Social e o desenvolvimento de jogos. Posteriormente, ao final deste capítulo, são apresentados o problema e o objetivo deste trabalho.

1.1 Psicologia Social

O estudo das ações de grandes grupos sempre foi o alvo da Psicologia Social [15], que busca realizar uma ponte entre a Psicologia e as Ciências Sociais. Existem diversas teorias na psicologia para explicar o comportamento humano, buscando responder perguntas como, "Assistir filmes violentos deixam as pessoas mais violentas?" ou "Escutar músicas de ritmo leve deixa as pessoas mais pacientes e tranquilas?". Para isso, testes a cerca de comportamentos humanos são realizados há muito tempo, mas nem sempre é possível chegar em resultados precisos.

O *General Aggression Model*, de Anderson e Bushman mostra que as ações, sejam elas pensadas ou impulsivas, de um indivíduo em um determinado momento é influenciado

por suas experiências vivenciadas recentemente (Figura 1.1). Mas não se pode esperar a mesma reação de diferentes pessoas ao vivenciar a mesma experiência, pois cada uma tem uma personalidade distinta. Uma mesma pessoa pode ter reações diferentes à um mesmo estímulo dependendo de seu estado atual. Em resumo, as ações são determinadas pela situação, pelo próprio indivíduo e e seu estado emocional no momento [13].

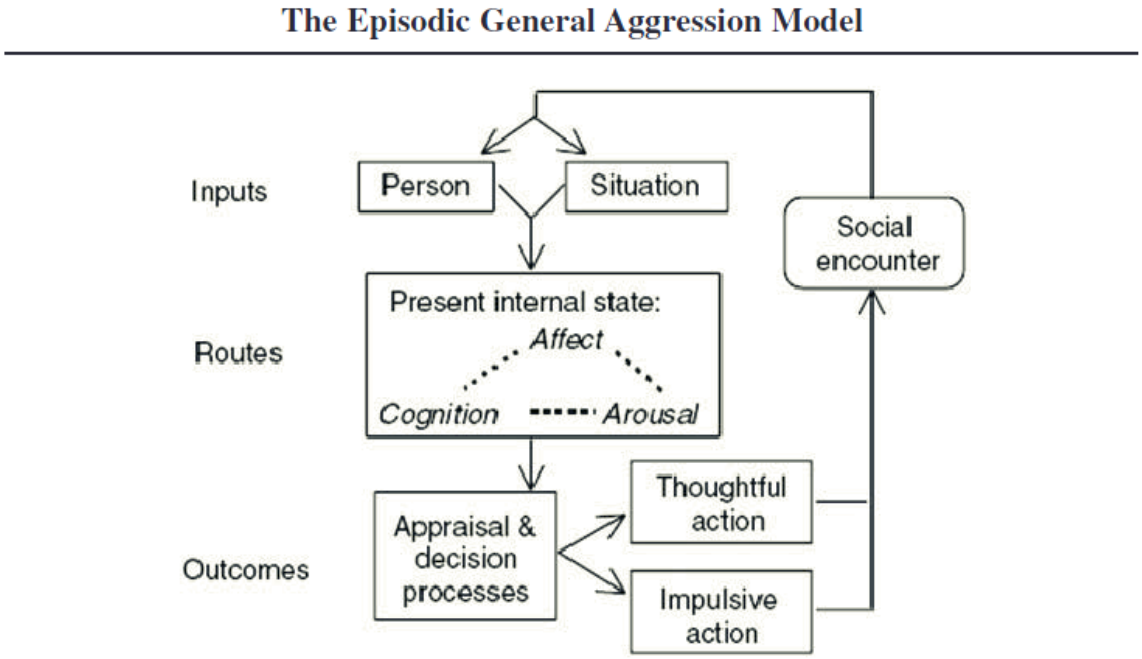


Figura 1.1: Esquema mostrando os fatores que afetam o comportamento [13].

Essa gama de variáveis que influenciam o comportamento do indivíduo torna desafiador a realização de pesquisas na área de comportamento. É necessário um grande número de voluntários na pesquisa para que seja reduzido ao máximo a variável da individualidade e se possa enxergar um padrão no comportamento de pelo menos a maioria dos participantes. A situação é o fator mais importante da pesquisa, devendo assim ser reproduzida com a maior similaridade possível entre os indivíduos, o que nem sempre é possível.

1.2 Desenvolvimento de jogos

Com o passar de tempo, jogos eletrônicos cada vez maiores e complexos estão surgindo. Atualmente, existem jogos capazes de suportar milhares de jogadores online simultaneamente, em mundos digitais tão grandes que pode se levar anos para explorá-los completamente. Para tornar isso possível, novas formas de desenvolvimento surgiram para facilitar a criação desses jogos.

O desenvolvimento de jogos digitais é um trabalho multidisciplinar que tem como base a computação e depende diretamente de artistas sonoros e visuais, além de outros profissionais, como designers, por exemplo, que possam ajudar a prover uma melhor experiência para o jogador. Atualmente, um jogo é um sistema altamente complexo e seu desenvolvimento completo requer a contribuição de diversas áreas da computação, como redes, engenharia de software, inteligencia artificial e computação gráfica.

A função principal de uma game *engine* é prover uma camada de abstração para que o desenvolvedor não precise se preocupar com detalhes da plataforma sendo utilizada. Assim, todas as *engines* provêm as seguintes funcionalidades:

- **Renderização**, seja ela 2D ou 3D.
- **Saída de áudio**, para música e efeitos sonoros.
- **Entrada de dados**, para controles ou outras formas do jogador interagir com o jogo.

Para jogos mais complexos que simulem comportamentos físicos de objetos ou que possibilitem diversos jogadores no mesmo jogo há a necessidade de *engines* mais complexas. Essas *engines* possuem funcionalidades adicionais que possibilitam:

- **Simulação de física**, para que o jogo possua um comportamento mais realístico.
- **Comunicação em rede**, para integrar diversos computadores em uma única seção de jogo.
- **Modelagem e animação**, para que todos os componentes 3D sejam perfeitamente integrados com o jogo.
- **Múltiplas plataformas**, desde consoles a celulares.
- **Entradas de dados alternativas**, como o *Xbox Kinetic*^{TM} ou *Playstation*®*Move*.
- **Suporte a scripting**, para um desenvolvimento mais fácil e ágil.
- **Ferramentas de depuração**, para facilitar a depuração de jogos mais complexos.

O código específico do jogo é desenvolvido em uma camada acima da *engine*. Ele utiliza todas as suas funcionalidades para implementar a lógica do jogo. O *design* do jogo é, em grande parte, independente da *engine* utilizada, mas, uma vez implementado o código, é totalmente dependente do sistema utilizado por ela. A *engine* utilizada define quais ferramentas estarão disponíveis para o programador assim como parte da arquitetura do sistema, como descrito por Freitas [22].

Quando se utiliza uma *engine* comercial, o código específico é implementado utilizando uma linguagem de *scripting*. Nela são feitas chamadas a todas as funções da *engine* necessárias para o código específico do jogo.

O uso de *scripts* é bem mais acessível e funciona bem para pequenas funcionalidades, mas não é eficiente. O principal problema dessa abordagem é que, por se tratar de um uso não previsto pelos desenvolvedores da *engine*, algumas partes do sistema podem não estar disponíveis por meio de *scripting*. Essa abordagem é muito utilizada por *modders*: pessoas que buscam fazer alterações em jogos comerciais, muitas vezes modificando os jogos a ponto de transforma-los um novo [23].

1.3 Caracterização do Problema

Para realizar as pesquisas de modo a reproduzir de maneira tão similar quanto possível a situação vivenciada pelo indivíduo, muitas estratégias são utilizadas. Porém, quando

esta pesquisa é feita com jogos, a situação foge do controle do pesquisador, pois a interação entre o computador e o jogo já foi pré-programada. Em alguns casos, para contornar esse problema, pesquisadores utilizaram-se de *mods* [21] de jogos para se adequar mais à necessidade da pesquisa, o que nem sempre é suficiente. Além disso, quando a pesquisa é realizada com jogos é comum ser necessário mais de um jogo para poder testar a situação que está sendo estudada e a situação oposta (um jogo violento e um jogo pró-social por exemplo) [13] e isso dificulta ainda mais manter as duas situações parecidas.

O Pró-Games [5], um grupo de pesquisa em Psicologia Social voltado a jogos, passa por estas mesmas dificuldades ao realizar suas pesquisas. O grupo atualmente utiliza jogos comerciais que não se adequam à suas necessidades e isso prejudica o resultado das pesquisas realizadas. Para um jogo se adequar a uma pesquisa em particular, ele deve ser produzido com este propósito. Ou seja, um jogo comercial, que tem, em geral, objetivos de agradar os jogadores para vender mais, dificilmente será ideal para uma pesquisa. Além disso, é importante coletar dados do jogador enquanto ele joga o jogo, como, por exemplo, a quantidade de vezes que ele morreu, o que normalmente não possível utilizando os jogos comerciais.

1.4 Objetivo

A proposta deste trabalho é desenvolver um jogo utilizando-se de técnicas de coleta de dados e de programação para jogos visando sanar o problema dos pesquisadores em Psicologia Social aplicada a jogos. É importante ressaltar que esse trabalho não irá realizar uma pesquisa de comportamento em si, a proposta é o desenvolvimento da ferramenta que será utilizada durante uma futura pesquisa. As principais características deste jogo são:

- **Diferentes modos:** o jogo apresenta um modo violento, um pró-social e um neutro;
- **Diferentes dificuldades:** cada modo possui um level design diferenciado para cada uma das três dificuldades: fácil, médio e difícil;
- **Homogeneidade:** o jogo é constante, ou seja, diferentes jogadores terão experiências semelhantes ao jogar;
- **Configurável:** o jogo é flexível, permitindo alterar vários elementos de modo a ser usado em diferentes pesquisas;
- **Coleta de dados:** o jogo registra as informações dos eventos que ocorrem durante a partida e armazena em arquivos de fácil acesso;
- **Dificuldade dinâmica:** o jogo pode controlar a dificuldade de acordo com o desempenho do jogador de modo a se adequar à jogadores completamente inexperientes ou muito experientes.

De modo a desenvolver este jogo com melhor qualidade, foram desenvolvidas ferramentas para auxiliar o level design, como, por exemplo, *scripts* para exportar terrenos do cenários desenhados no Photoshop® para *engine* Unity.

Com esse projeto é esperado que as pesquisas realizadas obtenham um resultado mais preciso e possa, assim, trazer para a sociedade uma resposta do real impacto de jogos.

1.5 Organização do documento

No Capítulo 2 é feita uma descrição e uma análise de diversos projetos realizados em Psicologia Social com jogos eletrônicos. É importante ressaltar que essa análise foi feita e discutida com os pesquisadores em psicologia social do grupo Pró-Games da UnB. Foi a partir dela que foram identificados os principais requisitos do jogo desenvolvido neste trabalho. Depois, ainda no Capítulo 2, são apresentados alguns jogos já desenvolvidos com um propósito diferenciado e que usam o métodos de coleta de informações a fim de mudar a experiência do jogador.

No Capítulo 3 é feita uma descrição detalhada do *game design* do jogo implementado. Durante a descrição são feitas breves explicações sobre as escolhas de alguns determinados elementos e como estes afetam a pesquisa realizada. O desenvolvimento e os detalhes de implementação são apresentados no Capítulo 4, onde é mostrado como foi usado o modelo de programação orientado a componentes, assim como as ferramentas desenvolvidas.

Ao final, foram realizados alguns testes com o jogo desenvolvido. O procedimento para a realização dos testes e os resultados são mostrados no Capítulo 5. No Capítulo 6, é apresentada a conclusão do trabalho e elementos que poderiam ser melhorados e adicionados, deixando assim uma referência para trabalhos futuros.

Capítulo 2

Trabalhos Correlatos

Este capítulo busca apresentar os trabalhos documentados correlatos ao tema da monografia. Como explicado no capítulo anterior o jogo proposto é uma ferramenta a ser utilizada em pesquisas de comportamento e por isso é realizado um levantamento de pesquisas de comportamento com jogos eletrônicos. Dentro deste levantamento são constatados alguns problemas envolvendo os jogos que são utilizados nestas pesquisas. Ao chegar na conclusão de que é necessário desenvolver um jogo exclusivo para este tipo de pesquisa, é feito um levantamento de outros jogos que também são desenvolvidos com um propósito científico.

2.1 Pesquisas de comportamento com jogos

Sempre foi motivo de pesquisa investigar se as atividades diárias das pessoas afetam seus comportamentos. Quando a televisão se popularizou, várias estudos foram conduzidos para descobrir se o conteúdo que era transmitido influenciava as ações dos telespectadores. Com videogames a preocupação é ainda maior. Ballard [12] afirma que os jogos eletrônicos, quando comparados com a televisão, necessitam de uma atenção maior do jogador, além de deixá-lo mais imerso no jogo devido ao fato de ser um passatempo mais ativo. Com isso em mente, são apresentadas abaixo várias pesquisas conduzidas no intuito de verificar a influência de videogames e jogos eletrônicos diversos no comportamento humano.

2.1.1 Pró-Games

Um grupo de pesquisa da Universidade de Brasília (UnB), chamado Pró-Games, vem pesquisando o efeito de jogos digitais no comportamento humano. O Grupo de pesquisa Pró-Games faz parte do GEPS (Grupo de Estudos e Pesquisa em Psicologia Social), fundado em 2007 na mesma universidade. O GEPS desenvolve atividades de pesquisa em Psicologia Social. A forma de trabalho do GEPS é integrada e colaborativa, sendo que cada integrante está diretamente vinculado a um projeto de pesquisa [5].

O grupo aponta que vários estudos evidenciaram a relação entre a exposição e uso continuado de jogos eletrônicos e diversas consequências para o indivíduo, como aquisição

de habilidades e conhecimentos, mudanças nos hábitos de jovens e adultos e mudanças comportamentais. Dentro deste último escopo, o foco principal de pesquisas tem versado sobre o comportamento agressivo. No entanto, evidências empíricas recentes sugerem que, a partir da presença de determinados elementos, os jogos eletrônicos podem estar relacionados a um aumento na ocorrência de comportamentos pró-sociais. Neste sentido, o presente projeto do grupo de pesquisa Pró-Games tem como objetivos: criar instrumentos de classificação de jogos eletrônicos (a partir da percepção de respondentes), verificar a influência de determinados jogos eletrônicos na ocorrência de comportamentos pró-sociais e investigar quais características do jogo influenciam a ocorrência de comportamentos agressivos e pró-sociais. Para atingir estes objetivos, foram delineados subprojetos que fazem uso de métodos correlacionais e experimentais de investigação. Está pesquisa ainda não está finalizada e está sendo coordenada pelo professor Mauricio Sarmet, doutorando e pesquisador do GEPS.

Abaixo é descrito como funciona o processo de coleta de dados na pesquisa que está sendo desenvolvida pelo grupo de pesquisa Pró-Games.

Funcionamento da pesquisa atual

Um voluntário é escolhido ao acaso de uma lista de voluntários cadastrados que é usada em diversas pesquisas na área da psicologia. Novos voluntários são recrutados dentro do campus e também podem ser convidados a participar da pesquisa. Ou seja, o público não é específico, apesar da maioria serem jovens de 19 à 25 anos conforme acontece nos ambientes universitários.

Não é dito ao voluntário o real objetivo da pesquisa, fazendo-os acreditar que trata-se apenas de um teste de um jogo. Este procedimento é conduzido dessa forma pois acredita-se que o fato do voluntário saber que o seu comportamento está sendo avaliado pode levá-lo a alterá-lo.

Cada usuário recebe um ID - para não ser identificado pelo nome - e responde a um questionário com perguntas pessoais como sexo e idade, além de perguntas sobre experiências passadas do sujeito como: “Com que frequência você costuma jogar jogos digitais?”.

Após o questionário ser respondido, é feito um sorteio para descobrir se o voluntário irá jogar o jogo violento ou não-violento. Nesta pesquisa é utilizado o mesmo jogo para ambas as situações, chamado “Fat Princess” (Figura 2.8), que possui uma mecânica similar a da brincadeira de pique-bandeira, onde dois times rivais devem resgatar sua princesa que está presa no castelo inimigo. Neste jogo o jogador pode desempenhar vários papéis para ajudar seu time, entre eles o de guerreiro, cuja função é invadir o castelo inimigo, matando quem está em seu caminho e resgatar a princesa. Outro papel é o do trabalhador, que tem a função de coletar recursos para melhorar as defesas de seu castelo e as armas de seu guerreiro. O sorteio é então usado para descobrir com qual destes papéis o voluntário irá jogar, ou o guerreiro (violento), ou o trabalhador (pró-social).

O jogador deve jogar por exatos 10 minutos. Após o jogo, é requerido preencher um questionário com percepções da experiência do voluntário a cerca do jogo, com perguntas como: “Você considerou o jogo violento?”, “De 1 a 10, qual seria o seu nível de diversão do jogo”. Em seguida, são feitas duas pesquisas para determinar as ações do jogador. Primeiro é encenada por outro pesquisador uma situação na qual a pessoa (o pesquisador)

entra na sala onde o participante está e, após pegar diversos materiais de cima da mesa, o ator demonstra que está com dificuldade para abrir a porta. Então é verificado se o voluntário tem o comportamento pró-social de ajudar o ator. O segundo teste requer que o voluntário complete uma história ambígua. A história começa sendo narrada com o relato do acidente de dois carros que se batem na estrada. O voluntário deve imaginar e escrever como os motoristas irão se comportar após esta colisão.

Ao final dos teste, é contado ao voluntário o real objetivo da pesquisa e entregue a ele um último questionário a cerca da própria pesquisa com perguntas como: "Você desconfiou do homem que veio buscar os materiais e precisou de ajuda para abrir a porta?". Todos os dados são digitalizados e armazenados em um computador para que sejam posteriormente agrupados e analisados por um outro software.

2.1.2 Agressividade em jogos de bilhar e jogos de luta

O experimento de Rose Wiest [12] teve como objetivo medir o nível de agressividade e reação cardiovascular dos participantes da pesquisa. Foram usados dois jogos, um considerado violento e outro não. O jogo violento foi o Mortal Kombat™ [12] (Figura 2.2), um jogo de luta e com ação intensa, e o jogo não violento foi o Corner Pocket™ [12] (Figura 2.1), um jogo de bilhar com estratégia e um pouco de ação. É importante notar que a pesquisa divide o Mortal Kombat™em dois jogos, pois dentro do jogo é possível configurar se este será muito violento (mostrando sangue, pessoas degoladas, entre outros elementos) ou menos violento.

Figura 2.1: Uma cena do jogo Corner Pocket™[12].

Seu método de pesquisa não foge do padrão. Cada voluntário, todos do sexo masculino, de 18 a 25 anos, jogaram um dos três jogos por 10 minutos. Enquanto jogavam, era medido seu batimento cardíaco. Depois era feita uma avaliação para detectar comportamentos agressivos advindos do voluntário.

Figura 2.2: Uma cena do jogo Mortal Kombat™[12].

Apesar dos dados apontarem que os jogos violentos realmente causaram comportamentos mais agressivos, Rose, explicitamente fala que o jogo Corner Pocket™e Mortal Kombat™ tem várias diferenças entre si, não somente o fator de violência, podendo estes outros fatores influenciarem nos resultados medidos. Além disso, foi possível observar que, pelas reações cardiovasculares, o jogo Corner Pocket™causava menos empolgação, fazendo com que o entretenimento de cada jogo seja outro fator que pode resultar em diferentes resultados.

2.1.3 Agressividade em jogos arcade e jogos de aventura

O experimento de Fleming [21] é muito semelhante ao experimento de Rose Wiest. Neste experimento, cujo foco foi em crianças na idade de 8 a 12 anos, os jogos foram escolhidos voltados para essa faixa etária. Ambos os jogos utilizados no experimento não apresentam violência intensa. O jogo não violento, Bouncer 2™ [21] (Figura 2.3), é um simples jogo de plataforma cujo objetivo é pular em diversas plataformas, esquivando-se de obstáculos e chegar ao topo. O Jogo violento chama-se Herc's Adventure™ [21] (Figura 2.4) no qual o jogador deve escolher um possível guerreiro (entre três diferentes) para começar uma aventura pela Grécia para salvar o mundo. Durante a aventura são enfrentados diversos inimigos, como esqueletos, ciclope e demônios que são encontrados no inferno.

Depois de conduzidas as pesquisas com crianças jogando um dos dois jogos, Fleming [21] chegou a conclusão que os jogos não estão relacionados com as atitudes dos jogadores logo após jogar. Contudo, na publicação dos resultados o grupo de pesquisa aponta algumas críticas com relação ao material escolhido para a condução dos experimentos. Dentre as quais, quatro são importantes ressaltar. Primeiro é que o jogo Bouncer 2™possui uma jogabilidade muito mais simples que o Herc's Adventure™. Isso fez com que crianças de idades menores e com pouca experiência em videogames tivessem uma experiência muito frustrante ao jogar este segundo jogo, algumas até mesmo desistiram no meio do jogo. Segundo é o fato de que o jogo violento tem um apelativo pró-social,

ou seja, seu objetivo final dentro do jogo era salvar o mundo e Fleming acredita que o ideal seria um jogo onde o personagem é malvado pelo simples fato de querer praticar atos de agressividade e não poderia ter uma motivação pró-social em segundo plano. Terceiro diz respeito ao fato do jogador poder escolher entre diferentes heróis no jogo Herc's Adventure^{TM}, um deles (uma arqueira) tem o poder de curar, o que foge ao tema agressivo, além disso seu estilo *cartoon* também interfere na percepção dos participantes. Quarto e último é o fato de que no jogo Bouncer 2^{TM}, alguns de seus obstáculos são bombas, o que interfere no caráter neutro do jogo, dando a este um tom de agressividade.

Figura 2.3: Uma cena do jogo Bouncer2^{TM}[21].

Figura 2.4: Uma cena do jogo Herc's Adventure^{TM}[21].

2.1.4 Testando estresse relacionado à música dos jogos

Sylbie [24] tem um objetivo diferente em sua pesquisa, mas no final chega à problemas semelhantes aos descritos até aqui. Nesta pesquisa o objetivo é descobrir se a música e efeitos sonoros de um jogo podem influenciar no stress do jogador, ou seja, é uma pesquisa mais voltada à saúde mental do que o comportamento psicológico. Para isso foi usado o jogo Quake 3™ [24] (Figura 2.5), um jogo de tiro em primeira pessoa (FPS ou *First Person Shooter*). Basicamente, o teste constitui em joga-lo com ou sem música. Antes e depois do jogo são medidas varias amostras do jogador (batimento cardíaco, amostras de saliva, entre outros). Com esses dados os pesquisadores chegaram a conclusão que jogar com sons aumenta o nível de estresse do jogador.

Semelhante às pesquisas anteriores, nesta pesquisa foi feita também uma crítica ao material utilizado. Sylbie explica que se o jogo Quake 3™tivesse a opção de escolher uma música calma ao invés da sua música agitada padrão poderiam ter sido coletados mais dados. O stress pode estar ligado também às diferentes armas e inimigos encontrados durante o jogo, já que estes produzem sons diferentes. Essa falta de uniformidade insere variáveis que podem interferir na interpretação dos resultados, fazendo com que o material utilizado perca parte de sua confiabilidade.

Figura 2.5: Uma cena do jogo Quake3™[24]

2.1.5 Situação Atual

Em 2009, Barllet [13] fez um levantamento sobre diversas pesquisas de comportamento com jogos eletrônicos datando de 1995 a 2008. Cada um destes estudos tem seu moderador , ou seja, o elemento que será alterado para se medir o comportamento, sendo eles: presença de sangue, violência, violência com "premiação" (quando o jogador recebe algo por cometer atos de violência), utilização de componentes interativos, competitivo contra cooperativo, ponto de visão (primeira ou terceira pessoa), sexo do personagem, interface do jogo, realismo e qualidade gráfica. Para cada um desses moderadores existem uma ou mais pesquisas, sendo que todas analisadas por Barllet não usam jogos produzidos

especialmente para esse fim. Isso implica dizer que, em sua maioria, as pesquisas feitas apresentam os mesmos problemas apontados nas seções anteriores sobre os experimentos de Rose, Fleming e Sylbie, mostrados anteriormente (Seções 2.1.2,2.1.3 e 2.1.4, respectivamente). Ou seja, nos jogos utilizados, além da variável analisada, dezenas de outras também são alteradas.

Em um caso ainda mais atual (2012), Bastian [14] fez uma outra pesquisa envolvendo o mesmo tema: jogos violentos. Neste trabalho, o autor afirma que jogos violentos são desumanizadores, no sentido de que os jogadores se tornam mais egoístas e apresentam poucas atitudes pró-sociais. Nessa pesquisa também foram usados dois jogos, um violento (Mortal Kombat™ 6 [14], Figura 2.6) e um não violento (Top Spin Tennis™ [14], Figura 2.7). Embora seja visível a superioridade gráfica destes para outros jogos, os problemas permanecem, isto é, há um conjunto de variáveis diferentes em cada jogo que mudam e influenciam fortemente a experiência do jogador.

Figura 2.6: Uma cena do jogo Mortal Kombat™6 [14]

Figura 2.7: Uma cena do jogo Top Spin Tennis™[14]

Recentemente, o grupo de pesquisa Pró-Games tem conduzido uma pesquisa com uma abordagem diferenciada, que utiliza o mesmo jogo como sendo violento e pró-social. O grupo utiliza o jogo Fat Princess™(Figura 2.8), um jogo online, onde até dezesseis jogadores se dividem em dois times para resgatar a sua respectiva princesa que está no castelo do time inimigo. Sua jogabilidade é bem simples, e, apesar de haver muita luta e sangue, seu estilo é bem cartunesco e infantil. Dentro do jogo existem diversas maneiras

de ajudar seu time a resgatar a princesa que vou reduzir aqui nas duas maneiras utilizadas nessa pesquisa: o guerreiro, que a missão é invadir o castelo inimigo (eliminando unidades da outra equipe) e resgatar a princesa; e o trabalhador, que a missão é coletar recursos para melhorar as defesas de seu castelo ou melhorar as armas e armaduras usadas pelos guerreiros. Os participantes da pesquisa são divididos entre aqueles que jogam como guerreiros (jogo violento) e aqueles que jogam como trabalhadores (jogo pró-social). O objetivo da pesquisa é medir o comportamento do jogadores depois de jogar.

O professor Mauricio Miranda Sarmet, coordenador da pesquisa do grupo Pró-Games, cita os pontos fracos que podem estar comprometendo o resultado da pesquisa. Primeiro, alguns jogadores não consideram o jogo como violento em nenhum dos casos. Ele acredita que isso se dá pelo fato de que o jogo tem um aspecto visual infantil. Segundo, o jogo pró-social está imerso em um cenário de violência, o trabalhador pode ser atacado pelos guerreiros inimigos. Terceiro, pelo fato do jogo ser online a dificuldade da partida não pode ser escolhida e diferentes jogadores podem ter experiências de jogo bem distintas. Quarto, o jogo não possui um limite fixo de tempo, sendo possível que uma partida acabe antes do tempo mínimo que o participante deve jogar.

Figura 2.8: Uma cena do jogo Fat Princess™ [3].

2.2 Análise dos trabalhos

Os relatos dos estudos apresentados neste capítulo possuem foco no processo de pesquisa propriamente dito. Concentram-se em apresentar o método utilizado para a coleta dos dados e os resultados obtidos utilizando um determinado método psicológico para averiguar a influência dos jogos eletrônicos no comportamento humano. Observou-se, entretanto, que os jogos utilizados não são objeto de análises e considerações mais aprofundadas. Acredita-se que isso esteja relacionado ao fato dos pesquisadores normalmente não terem acesso a recursos para desenvolver um jogo customizado para o experimento. Na prática, alternativamente, os estudos têm utilizado jogos disponíveis no mercado, que em sua maioria, se distanciam muito do jogo idealizado pelos pesquisadores ao começar uma pesquisa. Entretanto, esse fato tem influenciado negativamente os resultados, conforme relatado pelos próprios pesquisadores.

No âmbito das investigações que utilizam jogos distintos ou desenvolvidos com outros propósitos, com intuito de avaliar a influência dos jogos eletrônicos no comportamento humano, é possível elencar uma série de desafios:

- **Jogabilidade:** quando os dois jogos apresentam temáticas completamente distintas, como por exemplo, comparar o jogo de bilhar ao jogo de guerra;
- **Entretenimento:** os dois jogos tem diferentes níveis de entretenimento, que pode fazer com que o jogador ache mais divertido um jogo ao outro;
- **Imersão:** a capacidade do jogo de deixar o jogador imerso em sua história;
- **Dificuldade:** é o quão complicado o jogador acha determinado jogo. Jogos muito fáceis ou muito difíceis podem ser frustrantes;
- **Violência com conteúdo pró-social:** quando o jogo violento usado na pesquisa tem fatores pró-sociais, seja salvar alguém no final ou realizar ações de cura. O contrário também ocorre nestes casos, quando no jogo pró-social existem fatores violentos como inimigos, armas e sangue;
- **Diferentes experiências:** cada jogador, ao jogar, tem um experiência demasiadamente diferente. Isso pode acontecer devido ao jogo permitir ao jogador escolher caminhos diferentes, ou eles serem gerados aleatoriamente, ou mesmo os inimigos terem dificuldades variadas (como em um jogo online, onde inimigos são controlados por pessoas e por isso apresentam habilidades de jogo muito diferentes);
- **Visual:** os dois jogos tem estilos de arte muito distintos, ou então a qualidade visual de ambos muda significativamente;
- **Música:** os dois jogos diferem significantemente em seus efeitos sonoros e música de fundo;
- **Tempo:** quando o jogo, por algum motivo, limita o tempo de jogo (para mais ou para menos) fazendo com que os jogadores sendo avaliados joguem por períodos de tempo diferentes.

Um jogo comercial sem nenhum destes problemas é difícil de se encontrar, por isso, acredita-se que desenvolver um jogo com um propósito específico de auxiliar nesta pesquisas seja a melhor maneira de se aproximar de um jogo ideal. Este jogo deve permitir customizações, de modo a ser genérico suficiente para atender as demandas especificas de cada pesquisa.

Além dos desafios citados, a capacidade de armazenar e exportar as informações do *game play* do jogador também é um componente desejado. Os dados do *game play*, como por a pontuação do jogador, normalmente não podem ser exportados do jogo, mas acredita-se que estas informações podem ser usadas futuramente nas pesquisas.

2.3 Customização de Jogos

Ao instalar um jogo em seu computador, o usuário copia diversos arquivos contendo código compilado, áudio, imagens, entre outros. Esses arquivos definem como o jogo vai

se comportar. Nas primeiras gerações de jogos esse comportamento era bem definido, mas com o passar do tempo novas demandas foram surgindo e os jogos passaram a contemplar algumas que permitiram alterar o comportamento do jogo, tais como, os controles, nível de dificuldade ou outras interações que possam mudar a experiência do jogador.

Muitos entusiastas da área de jogos queriam uma experiência ainda mais diferenciada do que as opções existentes possibilitavam, assim eles passaram a explorar e alterar elementos do jogo. Naturalmente, a complexidade dessas alterações dependia muito da acessibilidade dos arquivos. Alguns arquivos, como texturas, eram muito mais acessíveis e intuitivos, já outros, como o comportamento de inimigos, muitas vezes necessitavam de engenharia reversa dos jogos.

Essas modificações não se limitam a jogos de computador, onde os arquivos são facilmente acessíveis: elas existiram mesmo nos jogos em cartuchos. Um exemplo de suceso de customização é "Ronaldinho Soccer 98"^{TM}uma modificação do jogo "International Superstar Soccer Deluxe"^{TM} [6], um jogo de futebol desenvolvido para SNES^{TM}que não previa nenhum tipo de modificação, que, ao se popularizar, chegou a ser comercializado no mercado sul-americano de jogos. O grande novidade implementada no jogo "Ronaldinho Soccer 98"^{TM}foi a possibilidade de jogar com times nacionais de futebol, o que tornou a experiência para jogadores brasileiros mais agradável. Para isso, não foi necessário fazer nenhuma alteração no código do jogo, bastando apenas alterar as imagens dos escudos dos times e o nome dos jogadores.

Figura 2.9: Uma cena do jogo Ronaldinho Soccer 98^{TM} [8]

2.3.1 Natureza das modificações

Uma das maiores preocupações dos desenvolvedores de jogos é prover uma experiência agradável ao jogador. Para um jogo comercial, essa experiência costuma ser contida

dentro de um escopo bem definido, sendo que as variações dentro do jogo são para que essa experiência agrade diferentes jogadores. Os fatores mais comuns que afetam essa experiência são nível de dificuldade e controles. Para alguns jogos específicos outros fatores se aplicam, como reduzir a violência ou apresentar sugestões de como prosseguir no jogo, por exemplo.

Pensando nesse crescente mercado e tendências, muitas empresas começaram a facilitar o desenvolvimento de modificações para seus jogos. Em alguns casos essas modificações se restringem a temas e valores como disponibilidade e dano causado por armas nos jogos. Em outros, são criados novos cenários, muitas vezes com mecânicas bem diferenciadas das do jogo original.

Um exemplo é o jogo Warcraft 3™[25], um jogo de estratégia em tempo real para o qual foi construído um grande suporte para modificações. Junto do jogo, existe uma ferramenta de edição de mapa muito completa que possibilita modificar diferentes aspectos do jogo, dentre eles, sons, modelos e comportamento dos personagens. Além disso, o serviço de jogos online permitem que os jogadores criarem sessões de jogos com qualquer mapa presente em seu computador. Isso possibilita aos criadores destas modificações divulgar e jogar com outras pessoas além de criar um mecanismo transparente de distribuição de jogos. Algumas modificações, como o Dota™[2], fizeram tanto sucesso, que se tornaram mais jogados do que o jogo original. Posteriormente foi desenvolvido um jogo exclusivamente baseado nessa modificação.

Muitas modificações como essa, são essencialmente feitas através de *engines* e alguns *assets* do jogo para gerar um novo sistema. Um exemplo de sistema foi criado por Harrop [23] para melhorar a visualização e interação com um sistema de administração de rede. Geralmente existe um limite até onde essas alterações podem chegar ainda mantendo um nível adequado de coerência para o usuário. Esse limite é definido pelo nível de acesso disponibilizado pelo desenvolvedor da *engine*.

2.4 Obtenção e uso de informações

Com a popularização dos jogos sociais, a obtenção de dados sobre o comportamento dos jogadores tem se tornado uma área de pesquisa valorizada pela indústria de jogos [42]. Esses comportamentos podem ser registrados de inúmeras maneiras, as quais devem ser cuidadosamente escolhidas levando em conta os objetivos desejados.

Isso pode ser visto, por exemplo, no caso de jogos *freemium*, cujo o objetivo é incentivar o jogador a comprar elementos vendidos dentro do jogo. Nesse caso, alguns dados que podem ser obtidos são os itens mais utilizados pelo jogador, sua faixa etária e frequência de jogo. Outras métricas podem facilitar diferentes objetivos e devem ser analisadas caso a caso.

Os dados coletados são computados na forma de estatísticas gerais dos usuários [43] e podem ser analisadas por *game designers* para tomar decisões sobre a evolução do jogo. Essas mesmas informações podem ser constantemente monitoradas para detectar problemas graves no jogo. Também podem ser geradas estatísticas utilizadas para estimar informações como o engajamento de cada jogador.

Processando individualmente os dados de cada usuário é possível prover uma experiência mais personalizada [11]. A customização automatizada da experiência do jogador é um trabalho complexo que depende muito da dinâmica e narrativa do jogo [30]. Outra

grande dificuldade para o tratamento individual é processar essas informações em tempo real, especialmente considerando que máquinas rodando jogos comerciais frequentemente têm seus recursos computacionais totalmente utilizados enquanto o jogo roda.

Um tipo de personalização simples é obtida ao comparar os dados individuais com as estatísticas gerais ou regras pré-definidas para o jogo. Usando essa abordagem alguns jogos geram mensagens personalizadas para os usuários como a mostrada na Figura 2.10.

Figura 2.10: Chamada do jogo Sim City Social [32] para jogadores que passam mais de 15 dias sem jogar.

2.4.1 Jogos de propósitos diferenciados

Uma forma de entretenimento muito encontrada em jogos é a solução de desafios. Tendo em mente a natureza humana de se interessar por desafios, pesquisadores criaram jogos que usam ações dos jogadores dentro do jogo para resolver problemas com grande complexidade computacional. Algumas pesquisas, como a de von Ahn [41], mostram diversas aplicações para esses problemas, tais como, treinamento de ferramentas especificas como a pesquisa de imagens do Google™, aplicações médicas, ou para treinar sistemas de inteligência artificial de diversos domínios.

Um desses jogos começou como projeto de uso de computadores domésticos para análise de enrolamento de proteínas. Por demanda de seus usuários se tornou um jogo [27] chamado Fold.it™, Figura 2.11. Essa *gamificação* do processo faz com que as pessoas

se voluntariem para ajudar na busca da solução, como explicado por Cooper [17]. Isso substitui o uso da força bruta pela máquina, um processo muito mais lento.

Figura 2.11: Logo do jogo Fold.it™

Vários outros jogos com um propósito estão disponíveis na página do projeto "Games With a Purpose" [4]. Entre eles o jogo Verbosity™(Figura 2.12) que tem como desafio ajudar um desconhecido a adivinhar uma palavra usando frases parcialmente definidas. A parte dessas frases que é preenchida pelo usuário é utilizada para treinar um sistema de linguagem natural.

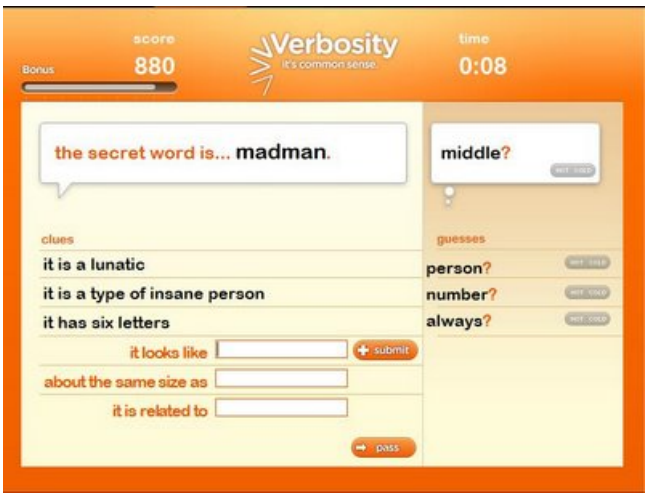


Figura 2.12: Jogo Verbosity™onde um dos jogadores tem que adivinhar palavras por meio de dicas.

Para utilizar esse tipo de pesquisa é importante garantir que as informações obtidas serão tão puras quanto possível. Ou seja, que os jogadores não utilizem respostas que prejudiquem o aprendizado. O descontinuado aplicativo Google Image Labeler™[41] sofreu muito com usuários que usavam palavras que se tornaram comuns entre os jogadores, mas que não tinham nenhuma relação com as imagens. Outros sistemas como o Fold.it™ submetem os melhores resultados para avaliação por especialistas e por isso não apresentam esse tipo de problema.

2.4.2 Uso dessas informações dentro do jogo

Ter informações sobre o comportamento do jogador pode ser útil para o próprio jogo. A principal aplicação disso é adaptar a inteligência artificial do jogo para prover uma

melhor experiência para o jogador. Na pesquisa de Spronck [34], esses dados refletem o resultado de combates entre a inteligência artificial e o jogador. Esses dados são então diretamente utilizados para alterar o comportamento dos inimigos.

Spronck compilou uma lista de quatro requisitos computacionais necessários para o desenvolvimento de uma inteligência artificial adaptativa. Sempre que aplicável, esses requisitos são utilizados neste trabalho como referência para o desenvolvimento do sistema de coleta de informações. Esses requisitos são:

- **Velocidade:** essencial para qualquer processamento que ocorre durante o *gameplay*;
- **Eficácia:** os dados coletados devem ser sempre aplicáveis. Para Charles [16], não podem ser utilizados dados que em algum momento levem a comportamentos absurdos;
- **Robustez:** tolerar condições aleatórias inerentes a jogos;
- **Eficiência:** em relação os estímulos utilizados para o aprendizado. O jogo deve ser capaz de tomar decisões mesmo quando existe carência de informações por parte dos jogadores.

Spronck também cita quatro requisitos funcionais considerados não-essenciais por não afetarem diretamente a experiência do jogador. Eles são importantes para os *designers* do jogo conseguirem trabalhar com liberdade e confiança. Eles são:

- **Clareza:** os resultados devem ser facilmente interpretáveis;
- **Variedade:** variedade de comportamento dos inimigos é importante para a experiência do jogador;
- **Consistência:** o tempo de aprendizado deve ser consistente entre diversas seções de jogo e para diferentes jogadores;
- **Escalabilidade:** a dificuldade deve aumentar de acordo com a habilidade do jogador.

Capítulo 3

Proposta

Considerando os pontos descritos nos capítulos anteriores, esse trabalho propõe criar um jogo digital para suprir as necessidades do grupo de pesquisa e resolver os problemas descritos pelos pesquisadores de psicologia social. O projeto foi baseado no jogo de tiro 2D estilo plataforma Metal Slug™ mostrado na Figura 3.1, onde o objetivo é chegar ao final do cenário enfrentando os diversos inimigos ao longo do caminho.

O jogo proposto é dividido em três partes principais, que são na verdade jogos independentes entre si. Esses três modos, que são escolhidos no início do jogo, representam: o jogo violento, o pró-social e o neutro. Apesar de independentes, os jogos compartilham entre si vários elementos em comum para preservar sua homogeneidade. O jogo conta ainda com uma ferramenta de dificuldade dinâmica que se baseia em um grande mecanismo de coleta de dados que podem ser inclusive exportados para futuras análises.

Figura 3.1: Capturas de tela do jogo Metal Slug Anthology™ [1]

3.1 Escopo

O escopo do jogo foi definido em conjunto com o grupo de pesquisa Pró-Games, levando em consideração a metodologia de pesquisa utilizada e as dificuldades encontradas pelo grupo. Nesse âmbito, foram estabelecidos plataformas, jogadores, gênero e objetivo do jogo.

Plataforma

O jogo foi projetado para executar nas seguintes plataformas:

- **Windows:** nativo e pelo navegador web;
- **Mac OS X:** nativo e pelo navegador web;
- **Linux:** nativo.

Essa liberdade de plataforma proporciona flexibilidade com relação aos equipamentos utilizados pelos pesquisadores. Além disso o uso do navegador web possibilita que algumas pesquisas mais simples e testes sejam feitos online, removendo um dos gargalos na coleta de dados, ou seja, o tempo.

Jogadores

O jogo é projetado para ser jogado por somente um jogador de cada vez, apesar de poder armazenar informações de diversos jogadores de forma separada. Isso permite aos pesquisadores coletarem as informações referentes aos voluntários de uma única vez ao final de uma bateria de voluntários, sem interromper o trabalho a cada novo voluntário.

Gênero

O jogo se encaixa dentro do gênero de "jogos de plataformas 2D". Esse tipo de jogo apresenta uma mecânica muito simples que permite jogadores casuais ou experientes jogá-lo sem dificuldade. Apesar de a mecânica ser 2D, é importante notar que a arte do jogo é 3D. Para cada modo de jogo (explicados na seção seguinte) existem sub-gêneros diferenciados.

Objetivo

A essência do jogo é que o jogador percorra a fase realizando ações que variam de acordo com o modo de jogo. Existe em todos os três modos uma pontuação geral e, para obter pontos, o jogador deve realizar as ações do determinado modo. Considera-se que o objetivo foi atingindo ao fim de um prazo definido pelo pesquisador (dez minutos, por exemplo) independente da performance do jogador. Ao fim desse tempo o jogo é encerrado e passa-se para outras etapas da pesquisa.

3.2 Mecânica do jogo

A mecânica do jogo é diferente para cada um dos três modos: Violento, Pró-social e Neutro. A principal preocupação ao projetar estes modos foi de passar ao jogador a uma experiência de violência, de pró-socialidade e de neutralidade (nem violenta, nem pró-social). Além disso, a mecânica deve se adaptar ao cenário, que é o mesmo nos três modos. Isso diminui a liberdade ao projetar cada modo, exigindo um maior esforço para sua criação.

3.2.1 Modo Violento

Este modo é o que mais se aproxima do jogo Metal Slug™. O jogo desse modo foi nomeado "The Killer" (Figura 3.2). O personagem principal é um soldado que tem por objetivo seguir sempre em frente matando inimigos diversos dotados de Inteligência Artificial (IA) simples. A única forma de obter pontos é matando estes inimigos, incentivando assim um comportamento mais violento. Este personagem executa seis ações: atirar com a arma básica, atirar com as armas especiais, usar uma faca, lançar granadas, pular e agachar.

O cenário que a ser percorrido é fixo (para cada dificuldade), ou seja, não existe caminhos alternativos ou aleatórios. O mesmo vale para os inimigos, isto é, são fixos. O jogo termina quando se esgota o tempo pré-determinado. Os inimigos do jogo são controlados por uma IA. Eles morrem ao levar tiros e, em sua maioria, atiram de forma lenta, o que possibilita ao jogador desviar dos tiros. Para tornar o jogo mais interessante, dois tipos de inimigos foram incluídos. Ao matar inimigos, armas especiais, kits de primeiros socorros ou caixas de granadas podem aparecer. Uma descrição mais detalhada dos inimigos e itens pode ser vista na Tabela 3.1 da Seção 3.3.

Figura 3.2: Jogo "The Killer".

A recompensa é um mecanismo utilizado na psicologia para estimular o indivíduo a fazer determinadas ações. Neste caso, o jogador é recompensado quando pratica atos violentos (matar inimigos), o que ajuda a determinar a experiência do jogador. Nos outros modos desenvolvidos, o jogador também é recompensado pelas ações que deve praticar.

O personagem tem dez pontos de vida que são reduzidos cada vez que ele é atingido, ou ao cair buracos. Quando os pontos chegam a zero ele morre. Após cair num buraco ou morrer o personagem recomeça em um lugar próximo de onde ele estava anteriormente e são reduzidos cem pontos de sua pontuação geral. Não existe uma quantidade máxima de vezes que o jogador pode morrer dentro do tempo determinado para o jogo durar.

3.2.2 Modo Pró-social

O personagem principal é um médico, cujo objetivo é encontrar soldados feridos espalhados pelo cenário. Este modo foi nomeado "The Help" (Figura 3.3).

Figura 3.3: Jogo “The Help”.

Existem desafios diferentes para salvar cada soldado e a velocidade com que se consegue realizá-los é o que determina o sucesso ou falha da cura do soldado. O tempo deste desafio é determinado pela dificuldade do jogo. Quanto mais soldados forem salvos ao final, maior será a pontuação do jogador.

A pontuação depende de quanto tempo o jogador levou para realizar um salvamento em relação ao tempo máximo permitido pela dificuldade. As seguintes mensagens (Figura 3.4) são mostradas ao jogador de acordo com o tempo restante para concluir o desafio:

- **“Good!”:** desafio concluído quando restar 0 a 25% do tempo limite;
- **“Great!”:** desafio concluído quando restar 25 a 25% do tempo limite;
- **“Awesome!”:** desafio concluído quando restar 50 a 50% do tempo limite;
- **“Perfect!”:** desafio concluído quando restar 75 a 75% do tempo limite.

Figura 3.4: Mensagens que aparecem ao jogador quando ele completa um desafio

Neste modo, existe outro tipo de soldado ferido que está caído no chão. Para salvar este soldado o jogador deve usar um sinalizador. Nesta parte do jogo existem dois desafios:

acertar o sinalizador no soldado e comandar o helicóptero para realizar o salvamento. O jogador também ganha pontos ao salvar este soldado de maneira similar à descrita anteriormente.

3.2.3 Modo Neutro

Ao escolher este modo, o jogador controla um explorador. O jogo desse modo foi nomeado "Treasure Hunt" (Figura 3.5). O objetivo neste modo é coletar moedas espalhadas pelo cenário. A jogabilidade é parecida com a do jogo Pró-social, porém, ao invés de encontrar soldados feridos o jogador encontra baús de tesouro que devem ser abertos concluindo um desafio.

Figura 3.5: Jogo "Treasure Hunt"

Ao abrir um baú, moedas aparecem e o jogador deve coletá-las. Estas moedas também são encontradas espalhadas pela fase e cabe ao jogador coletá-las explorando os diversos lugares do cenário. O jogador também recebe uma pontuação por abrir o baú com sucesso. Essa pontuação depende de quanto tempo o jogador levou para abrir o baú em relação ao tempo máximo permitido. As mesmas mensagens descritas no modo anterior (Figura 3.4) são mostradas de acordo com o tempo restante.

O personagem também possui um avião de brinquedo, que pode ser lançado. Esse avião ajuda a coletar as moedas do cenário que estão muito longe e pode também abrir um tipo de baú especial. Este é aberto quando o personagem ou o avião se aproximam dele, não sendo necessário vencer um desafio.

3.3 Elementos do jogo

Todos os elementos do jogo foram planejados de modo que o jogo atenda da melhor maneira possível as necessidades dos pesquisadores o utilizam como instrumento. Assim, vários aspectos que foram considerados normais para um jogo comercial são alterados para se adequar à pesquisa. As Tabelas 3.1, 3.2 e 3.3 mostram em detalhes os principais elementos de *gameplay* dos jogos violento, pró-social e neutro respectivamente.

Elemento	Detalhamento
	No jogo, o inimigo padrão só é ativado quando aparece na tela. Seu comportamento consiste em seguir o personagem, mantendo sempre um distância mínima. Sua velocidade de ataque (tiros por segundo) depende da dificuldade do jogo, assim como sua vida máxima. Esse inimigo sangra ao ser atingido, assim como o personagem. O inimigo também evita buracos no chão. Seu tiro tira um ponto de dano da vida do personagem.
	O inimigo de espada é uma variação do inimigo padrão. Ele realiza todas as ações que o inimigo padrão também realiza, porém não atira. Quando ele está próximo ao personagem ele usa sua espada. Cada golpe de espada atingido causa dois pontos de dano na vida do personagem.
	O kit de primeiros socorros é um dos itens que o inimigo pode deixar para o jogador ao morrer. Ele recupera um ponto da vida do personagem.
	A caixa de granadas é um dos itens que o inimigo pode deixar para o jogador ao morrer. O jogador, ao pegar a caixa, recebe três granadas, dentro do limite de dez granadas. O jogador pode as jogar granadas, que explodem depois de dois segundos matando imediatamente os inimigos próximos. Inimigos mortos pelas granadas têm seus corpos dilacerados.
	O lança-chamas é um dos itens que o inimigo pode deixar para o jogador ao morrer. Ao coletar um lança-chamas o personagem ganha cem tiros de lança-chamas e passa a equipar esta arma imediatamente. O lança-chamas, diferente do arma-padrão, atinge somente inimigos próximos ao jogador. Inflige três pontos de dano e pode ser usado através de paredes.
	A metralhadora é um dos itens que o inimigo pode deixar para o jogador ao morrer. Ao coletar a metralhadora o personagem ganha cem tiros de metralhadora e passa a equipar esta arma imediatamente. A metralhadora, diferentemente da arma-padrão, dispara quatro tiros ao ser usada. Estes tiros são mais rápidos, paralelos e com uma leve distância vertical entre si.

Tabela 3.1: Elementos do jogo “The Killer” (modo violento).

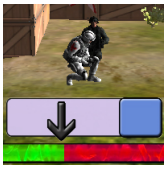
Elemento	Detalhamento
	Este é um soldado ferido que necessita ser salvo. Para este tipo de soldado o jogador deve apertar os botões na ordem que aparecem na tela, considerando o botão que está sobre a área destacada. Neste exemplo, o jogador deve pressionar: "X", "seta para a direita", "Z" e "X" para salvar o soldado.
	Este é um soldado ferido que necessita ser salvo. Para este tipo de soldado o jogador deve apertar o botão em destaque repetidas vezes. Cada vez que se aperta o botão, a imagem que representa o botão se aproxima da área destacada no painel. O jogo se encerra quando a imagem atinge a área destacada. Neste exemplo o jogador deve apertar a tecla "C".
	Este é um soldado ferido que necessita ser salvo. Para este tipo de soldado o jogador deve apertar o botão em destaque no momento que a imagem que representa o botão está sobre a área destacada. Se isso for executado corretamente a barra verde é incrementada. O jogo se encerra quando a barra é totalmente preenchida pela cor verde. Neste exemplo o jogador deve esperar a imagem do botão "seta para baixo" chegar à área destacada e apertar o botão "seta para baixo".
	Este tipo de soldado deve ser salvo com o uso do sinalizador. Quando o jogador usa um sinalizador próximo ao ferido o desafio com helicóptero é iniciado.
	O sinalizador pode ser jogado pelo personagem, persistindo no cenário por seis segundos até desaparecer. Não existe um limite de sinalizadores que o jogador pode usar, mas existe um tempo de espera mínimo de um segundo para o jogador poder usá-lo novamente.
	Quando o desafio com helicóptero se inicia o jogador perde controle do médico e passa a comandar o helicóptero. O helicóptero desce em direção ao ferido e deve ser alinhado sobre o ferido utilizando os teclas de "seta para esquerda" e "seta para direita".

Tabela 3.2: Elementos do jogo "The Help" (modo pró-social).

Elemento	Detalhamento
	Este é um tesouro que pode ser aberto pelo jogador. Neste tipo de tesouro o jogador deve apertar os botões na ordem que aparecem na tela, considerando o botão que está sobre a área destacada. Neste exemplo o jogador deve pressionar: "seta para direita", "seta para esquerda", "C", "seta para baixo" para abrir o baú.
	Este é um tesouro que pode ser aberto pelo jogador. Neste tipo de tesouro o jogador deve apertar o botão em destaque repetidas vezes. Cada vez que se aperta o botão, a figura do botão se aproxima da área destacada. O jogo encerra quando a figura atinge a área destacada. Neste exemplo o jogador deve apertar o botão "X".
	Este é um tesouro que pode ser aberto pelo jogador. Neste tipo de tesouro o jogador deve apertar o botão em destaque no momento que este está sobre a área destacada. Se isso for executado com corretamente a barra verde é incrementada. O jogo se encerra quando a barra é totalmente preenchida pela cor verde. Neste exemplo o jogador deve esperar a imagem da tecla "Z" chegar à área destacada e apertá-la.
	O explorador possui um avião de brinquedo que fica circulando em torno dele. Este avião, quando arremessado, ajuda a recolher as moedas espalhadas no cenário, alcançando moedas inalcançáveis pelo jogador de per se. Ao arremessar o avião, este avança para frente e para cima e depois volta em direção ao jogador carregando as moedas coletadas.
	Este tipo de baú tem um desenho de avião e está espalhado pelo cenário. O jogador não precisa executar nenhuma ação para abrí-lo, basta encostar nele. O avião de brinquedo também pode abrir este baú.
	Moedas devem ser coletadas pelo jogador e com isso são concedidos a ele dez pontos. Existem dois tipos de moedas. O primeiro tipo são moedas encontradas flutuando pelo cenário. O segundo tipo são moedas que saem de dentro do baú de tesouros. Esse segundo tipo de moeda é físico, ou seja as moedas sofrem ação da gravidade e caem no chão. Este tipo de moeda não é coletada pelo avião de brinquedo.

Tabela 3.3: Elementos do jogo "Treasure Hunter" (modo neutro).

Independente do modo, o cenário é o mesmo sempre que o jogador iniciar um novo jogo, e foi planejado de modo que o jogador não consiga finalizá-lo em menos de dez minutos, ou seja, o jogador não irá chegar ao final do cenário independente do tempo de jogo escolhido nas configurações, haja visto que o limite deste tempo que pode ser configurado é de 10 minutos. O requisito de homogeneidade entre os modos foi decisivo na definição do mesmo cenário para os três modos. Foi escolhido um cenário mais neutro possível para não influenciar nenhum dos modos de maneira negativa ou positiva.

Esse cenário é divido em fases para não sobrecarregar o jogo. Ao chegar no final de uma fase a próxima é carregada. Foram construídas o cinco fases no total (Figura 3.6). Essas fases foram projetadas e montadas por um voluntário com experiência em *level design*.

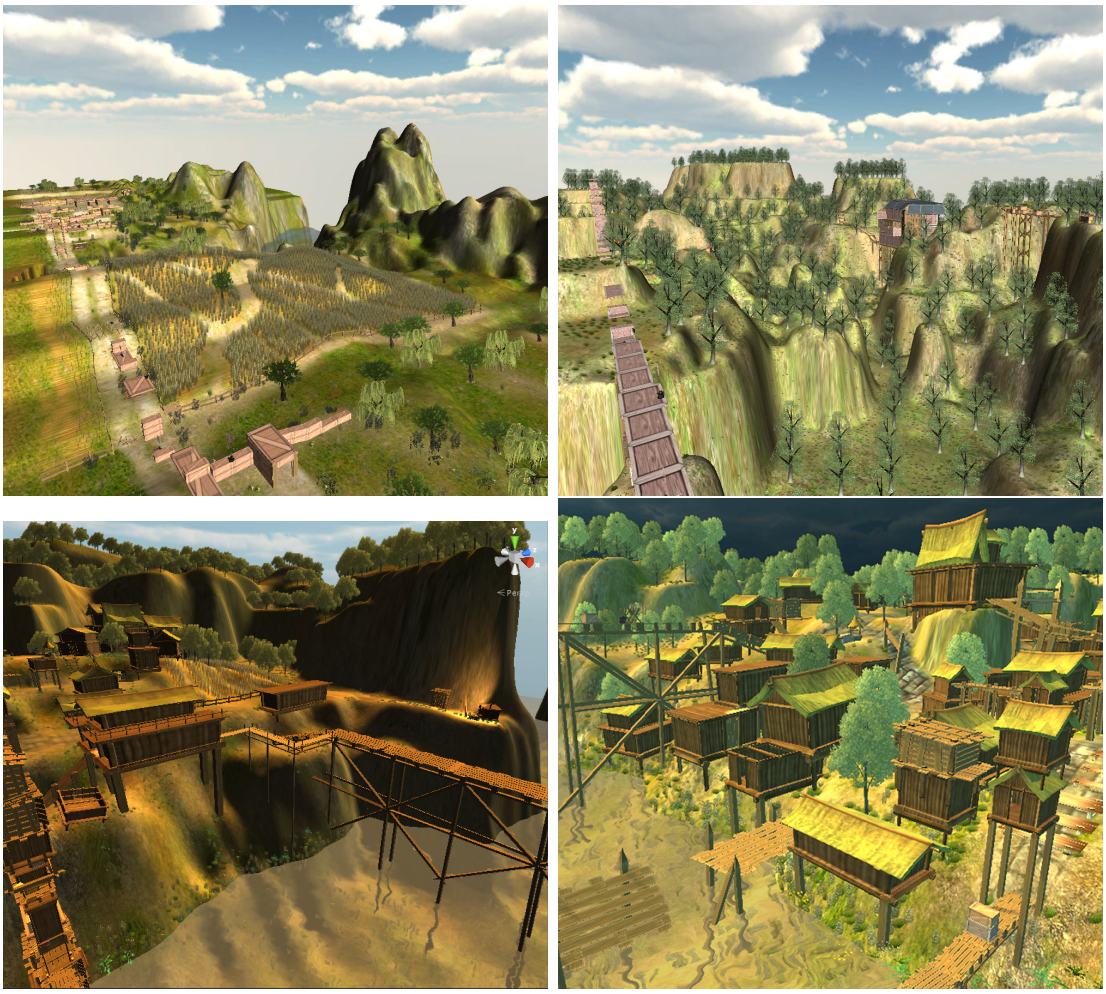


Figura 3.6: Fases do jogo

Antes da primeira fase existe um tutorial, que não contém inimigos, soldados para serem salvos ou qualquer outro elemento de algum modo específico. Neste tutorial existem placas no fundo do cenário falando o que o jogador deve fazer e alvos que representam os elementos dos modos. O jogador, pode interromper o tutorial quando julga que já aprendeu o suficiente para iniciar o jogo. O tempo gasto neste tutorial não é contabilizado

no tempo total da pesquisa. O objetivo do tutorial é previamente adaptar o jogador aos controles do jogo, sem expor aos efeitos específicos de cada modo.

Os controles foram pensados para que a maioria dos participantes possa jogar o jogo sem dificuldades, ou seja, com controles mais simples possíveis. Também foi tomado o cuidado para que todos os modos apresentassem a mesma dificuldade e não se tornassem frustrantes para o jogador. Isso foi alcançado criando jogabilidades parecidas em todos os modos de jogo. Os modos pró-social e neutro são os que mais se aproximam em questão de jogabilidade se diferenciando somente do modo violento. Porém, foi discutido junto aos pesquisadores que o ato de atirar e desviar de tiros, na perspectiva do *gameplay*, são ações válidas para se comparar com o ato de abrir baús e salvar pessoas.

O menu do início do jogo permite ao jogador escolher entre os três modos de jogo. Tais botões, para selecionar o modo de jogo, são nomeados como "modo1", "modo2" e "modo3". Isso acontece pois os participantes da pesquisa não podem ter qualquer conhecimento a cerca do real intuito da pesquisa até que esta seja finalizada. Colocar nomes como "modo violento", "modo pró-social" e "modo neutro" poderia sugerir ao participante o real objetivo da pesquisa.

O menu de configurações do jogo não é visível ao jogador, por isso deve ser pressionado um atalho no teclado para ele ser exibido. Assim, somente os pesquisadores podem alterar as configurações. Além disso, nenhuma dessas configurações é apresentada ao jogador.

No menu de configurações, mostrado na Figura 3.7, são determinadas quatro opções:

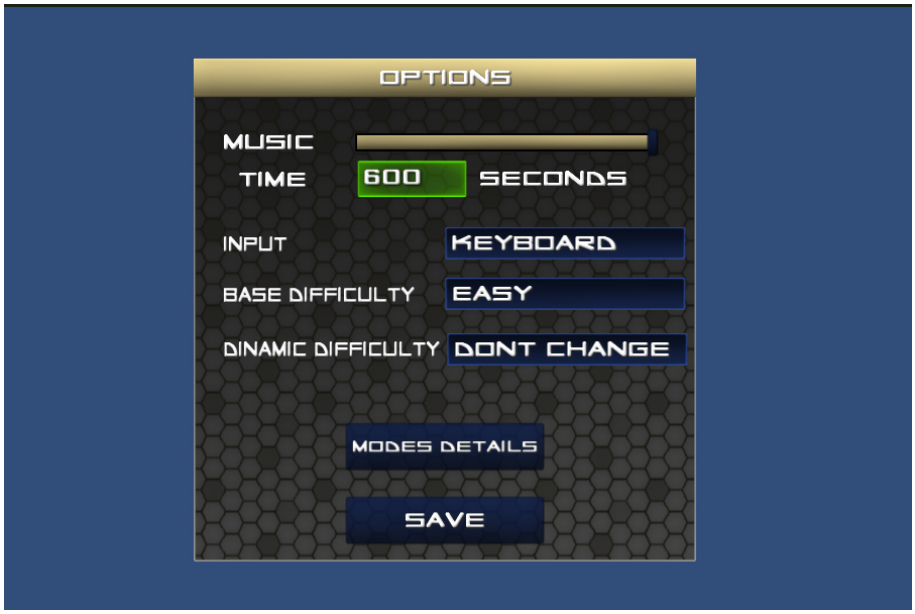


Figura 3.7: Menu de configurações do jogo

- **Music**: determina a altura da música de fundo do jogo;
- **Input**: determina se o jogo será jogado com o teclado ou com o *gamepad*;
- **Base Difficulty**: determina a dificuldade base do jogo. É explicado com mais detalhes na Seção 3.5;

- **Dynamic Difficulty**: determina a dificuldade dinâmica do jogo (detalhes são apresentados na Seção 3.5).

Além do menu de configurações principal, existe um segundo menu para controlar algumas variáveis presentes no jogo. Cada uma delas pode ser aplicada separadamente para cada modo de jogo, porém algumas variáveis não se aplicam a todos os modos:

- **Sangue nos inimigos:** no modo violento, determina a presença ou ausência de efeitos de sangue ao acertar um inimigo com tiros. No modo pró-social, determina este efeito nos soldados feridos. Esse item não se aplica ao modo neutro;
- **Sangue no personagem:** no modo violento, determina a presença ou ausência de efeitos de sangue quando um inimigo acerta o personagem ou quando este cai de grandes alturas. No modo pró-social e neutro, este efeito só acontece quando o personagem cai de grandes alturas;
- **Fogo no cenário:** determina a presença ou ausência de fogo como elemento do cenário (queimando casas e árvores) em todos os modos;
- **Fogo nos inimigos:** determina, no modo violento, se o inimigo irá pegar fogo ao ser acertado por lança chamas ou granadas. No modo pró-social determina se irá aparecer soldados pegando fogo. Esse item não se aplica ao modo neutro.

A música neutra do jogo também foi discutida com o grupo de pesquisa. Foi necessário criar uma música neutra, haja visto que esta se repete nos três modos de jogo. Ao contrário dos efeitos sonoros que são diferentes em cada modo. Para isso foi solicitado a ajuda de um experiente *sound designer* que se voluntariou para desenvolver sons e música de acordo com a experiência desejada.

Na HUD (Head Up Display) estão dispostos os elementos que fornecem informações ao jogador. Em todos os modos temos um *display* mostrando a pontuação e o tempo que falta para o jogo encerrar. Além disso, é mostrado, de maneira intuitiva, as ações que o jogador pode realizar com o personagem naquele instante. Para o modo violento existe ainda uma barra de vida mostrando a vida atual do personagem em cor verde e um *display* mostrando a munição atual da arma que se está carregando e a quantidade de granadas disponíveis.

3.4 Coleta de Dados

Ao iniciar um jogo, o pesquisador deve informar um ID que irá identificar o voluntário que está jogando. O controle desses identificadores é feito pelo próprio grupo de pesquisa. Durante o jogo são armazenados diversos dados do jogador e suas ações.

O jogo permite então exportar os dados de cada voluntário para arquivos em formato `.csv` para que sejam lidos e analisados por programas externos. Além disso, o jogo permite agrupar estes arquivos por data ou modo jogado para uma melhor organização. Esta opção para gerar arquivo está no menu que fica escondido do jogador.

Estes arquivos contém dados que, em sua maioria, são impossíveis de serem obtidos em um jogo comercial. Ao se aplicar à programas de externos voltados à análise de dados brutos acredita-se que possa ser encontrado alguma relação destes dados com dados de outras etapas da pesquisa, possibilitando a produção de novos conhecimentos.

3.5 Dificuldade Dinâmica

A coleta de dados que é feita durante o jogo também é usada para modificar a experiência do jogador. No menu de opções é possível determinar se a dificuldade do jogo será dinâmica ou estática. Para implementar a dificuldade dinâmica é necessário monitorar, coletar e organizar todos os dados que indicam o desempenho e o comportamento do jogador durante o jogo.

A dificuldade estática do jogo pode ser escolhida no menu de opções dentre três possibilidades: "fácil", "médio"ou "difícil". Essa dificuldade modifica, no modo violento, a inteligência artificial dos inimigos, quantidade de vida destes e a frequência com que o kit de primeiros socorros e armas especiais são fornecidas ao jogador. No modo pró-social e neutro a dificuldade afeta o tempo máximo para se obter sucesso ao abrir um baú ou salvar um soldado.

A dificuldade dinâmica também possui três opções: “desligada”, “sempre fácil” e “sempre difícil”. É importante que a opção de dificuldade dinâmica esteja desligada nas situações em que a pesquisa não leve em consideração o nível de frustração do jogador. Porém, pode ser interessante medir se um jogo de dificuldade adaptada, seja por ser muito fácil ou difícil, irá afetar o comportamento do jogador quando comparado ao jogo no nível de dificuldade “médio”, por exemplo. Para esse tipo de hipótese foram criadas as opções “sempre fácil” e “sempre difícil”. Nessas opções de dificuldade o jogo vai se adequando ao jogador para colocá-lo em uma dificuldade relativa ao quão bem o jogador esta se saindo. Essa dificuldade pode até ser mais fácil que o nível estático “fácil” ou mais difícil que o nível estático “difícil”. Não é interessante criar níveis estáticos “muito fácil” ou “muito difícil”, pois corre-se o risco do jogo perder sua dinâmica. Um jogador muito inexperiente, por exemplo, pode ficar tão preso em um suposto modo “muito difícil” que nem sequer irá matar um inimigo. Esse tipo de extremo não seria útil para o pesquisa e descaracterizaria o jogo.

Mesmo o jogo tendo sido configurado para que exista uma dificuldade dinâmica, a dificuldade estática ainda influencia no jogo, sendo ela a inicialmente pré-estabelecida. A dificuldade dinâmica muda os mesmos aspectos que a dificuldade estática, mas pode extrapolar estes aspectos de modo que alguns deles fiquem mais fortes que outros. Por exemplo, pode-se aumentar a vida do inimigo sem alterar a quantidade de itens que ele deixa ao morrer. Estas decisões são feitas de acordo com os dados coletados.

A Tabela 3.4 mostra como uma parte do cenário, neste caso um buraco no meio do terreno, pode sofrer variações de acordo com a dificuldade atual do jogo. Essas variações não são no terreno do jogo em si, pois este é sempre constante, mas sim, nos elementos do cenário que podem facilitar ou dificultar determinada ação. Vale destacar que no jogo de dificuldade dinâmica o cenário também é modificado dinamicamente mas sem o jogador perceber.

Cenário	Detalhamento
	Este é o cenário do jogo na dificuldade fácil. É colocado uma ponte sobre o penhasco para que seja impossível o jogador cair. Além da dificuldade dos inimigos ser pequena, eles também aparecem em pouca quantidade.
	Este é o cenário do jogo na dificuldade média. A ponte sobre o penhasco é retirada e é possível o jogador cair. É colocada uma caixa antes da ponte de modo a facilitar o pulo realizado pelo personagem, pois nessa situação ele deve pular de um lugar mais alto para um mais baixo. Nesse cenário existem mais inimigos.
	Este é o cenário do jogo na dificuldade difícil. A ponte sobre o penhasco é retirada e é possível o jogador cair. São colocadas caixas logo após o penhasco de modo a formar uma parede alta. Essa parede dificulta o pulo, obrigando o jogador pular de um lugar mais baixo para um lugar mais alto. Neste cenário existem muitos inimigos e eles são colocados em diferentes plataformas o que dificulta serem atingidos pelo jogador.

Tabela 3.4: Diferentes dificuldades do jogo.

Capítulo 4

Implementação

Este capítulo descreve o processo de implementação do jogo proposto neste trabalho e das ferramentas que foram desenvolvidas ao longo do processo. Primeiramente, são discutidas as etapas que antecederam o desenvolvimento do jogo de per se, como a criação de um protótipo e ferramentas externas necessárias. Em seguida, são apresentadas extensões ao editor da Unity [37] criadas para possibilitar o desenvolvimento do sistema de coleta e tratamento de dados. Por fim, é explicada a integração desses componentes durante a execução do jogo.

O jogo foi desenvolvido usando a Unity, uma *game engine* cuja a programação é orientada a componentes. Cada objeto usado no jogo tem atribuído a ele um ou mais componentes. Existem diversos componentes que são disponibilizados pela própria Unity para realizar as funções mais básicas, como por exemplo o `Collider` que controla colisões. Porém, é necessário construir componentes específicos do jogo como o componente `AbreBau` que permite o objeto abrir os baús de tesouro. Assim, um mesmo componente pode ser usado em diversos objetos como foi o caso do `MetalSlugController` que realiza o controle do personagem principal dos três modos desenvolvidos. Em [22] Freitas, é mostrado como a programação orientada a componentes é utilizada em diversas *engines*. Esse paradigma reduz a quantidade de código duplicado e facilita o desenvolvimento paralelo, permitindo várias pessoas trabalharem sobre o mesmo objeto, produzindo diferentes componentes.

Ao longo deste capítulo são feitas diversas referências aos eixos X, Y e Z, ilustrados na Figura 4.1.

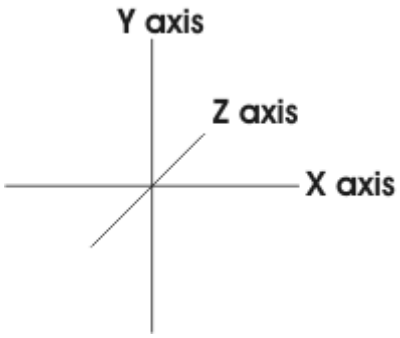


Figura 4.1: Eixos X, Y e Z.

Utilizando a convenção adotada pela Unity, esses eixos são:

- X: crescente para direita;
- Y: crescente para cima; e
- Z: crescente para o fundo do cenário, em direção ao horizonte.

4.1 Projeto

Inicialmente planejou-se desenvolver o jogo 2D, semelhante ao jogo Metal Slug™. Seguindo essa ideia foi desenvolvido um protótipo 2D com a mecânica básica para esse estilo de jogo: andar e atirar. Com base nesse protótipo e na análise de outros jogos de plataforma foi possível realizar diversas considerações que influenciaram na decisão final de implementar um jogo em 3D.

4.1.1 Protótipo inicial

Em jogos de plataforma 2D a profundidade do cenário é simulada utilizando o efeito de *parallax*. Esse efeito consiste em mover camadas em velocidades distintas a medida que o jogador se desloca no eixo X. Camadas mais próximas se deslocam mais rapidamente para dar a ilusão de estarem mais próximas ao jogador, como a camada com contorno azul da Figura 4.2. Isso faz com que o jogador tenha uma noção de profundidade razoavelmente realística com relação ao ambiente em geral.

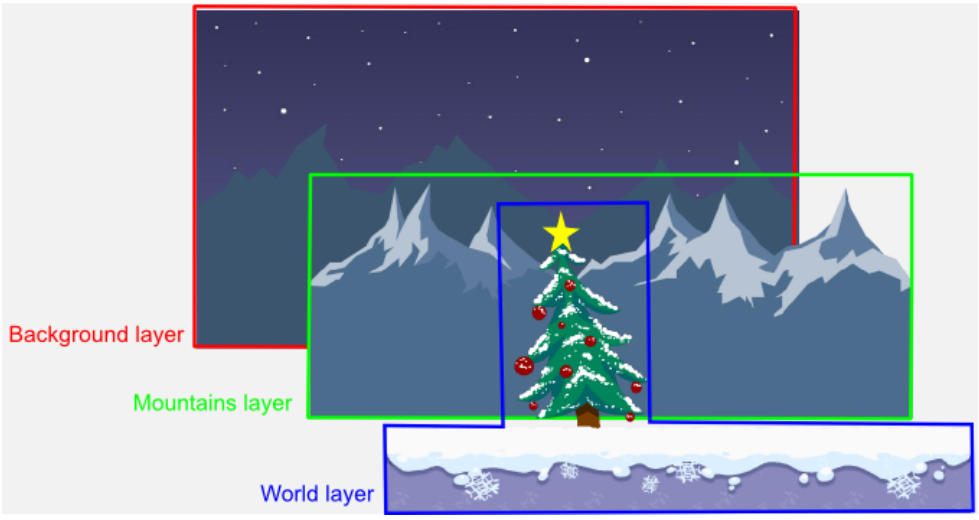


Figura 4.2: *Parallax* divida em três camadas: plano de fundo, montanhas e principal.

Utilizando esse efeito, qualquer plataforma no plano principal tem uma única posição no eixo Z. Isso possibilita liberdade no posicionamento de plataformas, bastando que a profundida seja indicada pelo traço da arte do plano principal, como mostrado no exemplo da Figura 4.3 onde é possível ver três níveis de altura/profundidade representados em um único plano. Assim, ao se mover no eixo Y o avatar utiliza diferentes plataformas sem que ele se desloque no eixo Z.

Figura 4.3: Camada principal do jogo Metal Slug Anthology™ [1].

É importante ressaltar que em um jogo 2D toda arte é construída utilizando *sprites* e *sprite sheets*, que são várias imagens que, se mostradas em sequência, formam a animação desejada. Cenários e animações construídas com *sprites* requerem um trabalho artístico muito grande. Essa demanda foi um dos problemas enfrentados para dar continuidade ao protótipo inicial em 2D.

4.1.2 Versão 3D

A versão 3D do jogo implementado utiliza uma câmera com perspectiva que se move junto com o personagem. Todos objetos do cenário são posicionados no espaço tridimensional, criando um efeito muito mais realístico quando a câmera percorre o cenário.

Por outro, lado essa perspectiva dificulta o uso de diferentes níveis. Na Figura 4.3, por exemplo, cada nível teria uma profundidade diferente. Neste estudo, é imperativo manter a jogabilidade simples e intuitiva. Por isso buscou-se criar um cenário onde todas as plataformas estão alinhadas com o plano onde o jogador se move, mesmo que isso crie construções claramente surreais, como por exemplo pontes de maneira flutuando no ar.

Esse estilo de plataformas também afeta os controles do jogo. Para se tornar mais intuitivo para jogador interagir com as plataformas e se movimentar entre elas diretamente, foram feitas algumas decisões:

- o personagem não atravessa plataformas;
- o personagem não mira para cima ou para baixo;
- o cenário deve incentivar desafios que envolvam dar voltas em obstáculos.

Essas decisões contribuem para uniformização da jogabilidade entre os diferentes modos de jogo. Além disso, a opção pelo jogo 3D reduziu a necessidade de um artista para

ajudar no desenvolvimento do jogo. O cenário foi construído utilizando a própria ferramenta de criação de terreno da Unity. Modelos e animações genéricas, como pessoas e movimentação das mesmas, podem ser encontrados à venda na internet. A união destes modelos e animações foi feita utilizando outra ferramenta da Untiy, a Mecanim, apresenta a seguir na Seção 4.2.2. O ambiente 3D ainda permite agrupar elementos para criar animações mais complexas, como por exemplo o sangue, que foi feito separadamente e depois aplicado ao personagem para possibilitar a fácil remoção do mesmo. Em um jogo 2D, neste exemplo do sangue, seria necessário a criação de duas animações para cada personagem ferido, isto é, com sangue e sem sangue.

4.1.3 Terreno

Todos os objetos e personagens do jogo precisam estar distribuídos sobre o terreno. A maneira mais indicada de se criar esse tipo de superfície é com a ferramenta de terreno da Unity [39]. Essa ferramenta utiliza mapas de altura para criar um terreno visualmente atraente e otimizado.

A Unity provê um editor visual para criar terrenos de maneira orgânica, voltada para jogos com movimentação num ambiente 3D. Porém, em jogos clássicos de plataforma o chão possui características bem artificiais (formas bem definidas) devido a visão lateral adotada nele. Ao passar para o ambiente 3D é importante não perder essa simplicidade e precisão. Para possibilitar um *design* pensando em plataformas 2D foi criado, neste trabalho, um *script* para possibilitar a criação do terreno utilizando uma ferramenta gráfica como o Adobe®Photoshop®. Esse *script* é melhor explicado na Seção 4.2.3.

4.2 Ferramentas Utilizadas

Considerando a natureza multidisciplinar de jogos e as considerações feitas na seção anterior, é natural que diferentes ferramentas sejam utilizadas durante a produção de um jogo. Nesta seção são descritas algumas ferramentas que foram essenciais no desenvolvimento desde trabalho.

4.2.1 Modelagem e animação 3D

Uma vez decidido desenvolver o jogo em um ambiente 3D, é necessário pensar nos modelos e animações que serão utilizados. Para isso, esses modelos devem estar no formato Autodesk FBX [10].

O primeiro passo consiste em obter os modelos dos personagens. Assim, foram procurados modelos gratuitos online, como por exemplo os soldados disponíveis na Unity Asset Store [38] mostrados na Figura 4.4. Em alguns modelos, como o do médico, foram feitas alterações nas texturas, utilizando uma ferramenta de edição de imagem.

Estes modelos já possuem uma estrutura de ossos e músculos que possibilita a Unity integrá-los com as animações. Por isso, foi adquirido um conjunto de animações para serem utilizadas com esses modelos, conforme é descrito na Seção 4.2.2 a seguir.

Figura 4.4: Modelos 3D dos soldados utilizados no jogo, obtidos na Asset Store [29].

4.2.2 *Mecanim*

A animação de personagens 3D é mais trabalhosa que a própria modelagem. Por isso, foi adquirida uma seleção de animações básicas para os personagens do jogo. Essas animações precisam ser definidas de acordo com o personagem e podem ser diretamente controladas pelo código do jogo. Para isso, foi utilizada a ferramenta *Mecanim*. Ela consiste de um sistema integrado à Unity a partir da versão 4.0 [37] que possibilita excelente transição e combinação das animações. Graças a essa ferramenta não foi necessário fazer alterações diretamente nas animações, apenas a mistura (ou *blending*) das animações do pacote adquirido.

Contudo essa seleção não continha todas as animações específicas para o jogo. Assim, foi necessário separar componentes de animação em diferentes camadas e controlá-los a partir do código que gerencia as ações do personagem. Isso é facilitado por um componente genérico desenvolvido para ativar e desativar animações de acordo com a necessidade. Esse componente serve para:

- aplicar certas animações específicas a alguns membros do corpo, como no arremesso de granadas;
- substituir posições indesejadas em animações como a de agachar; ou
- aplicar animações aleatórias, a fim de criar uma impressão de movimentos mais naturais.

4.2.3 Photoshop®

Conforme descrito anteriormente a ferramenta nativa da Unity para edição de terrenos não é adequada para prover a experiência de um jogo de plataforma. Para contornar suas limitações essa ferramenta é capaz de exportar/importar arquivos como o mapa de alturas do terreno com qualquer relevo desejado. O relevo do terreno é tratado como uma malha retangular composta de triângulos, como mostrado na Figura 4.5. Dois triângulos formam um quadrado no terreno. Como cada vértice do quadrado pode estar em uma altura diferente, é possível criar diversos tipos de inclinação no terreno, com certas limitações.

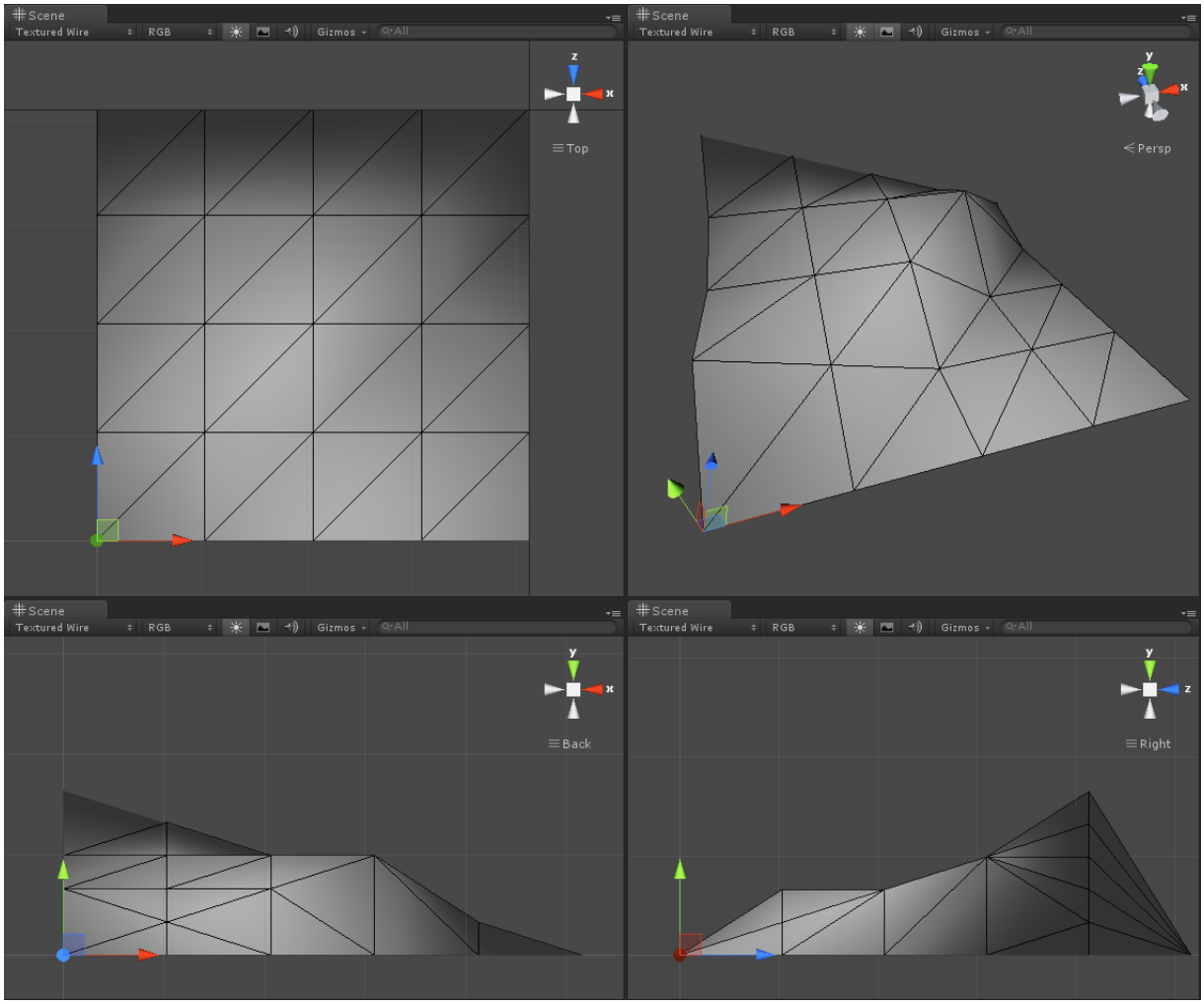


Figura 4.5: Exemplo simplificado de terreno utilizando malha triangular com trinta e dois triângulos formados por vinte e cinco vértices.

A representação utilizada para exportar esse mapa é de uma imagem onde cada vértice é representado por um pixel e sua altura representada pela cor do pixel. O terreno formado por trinta e dois triângulos mostrado na Figura 4.5 é representado por uma imagem quadrada com vinte e cinco pixels, mostrada na Figura 4.6. Por meio deste exemplo é possível ver onde o pixel mais claro no canto superior direito representa o pico e a parte totalmente preta no canto inferior direito representa a parte plana mais baixa.

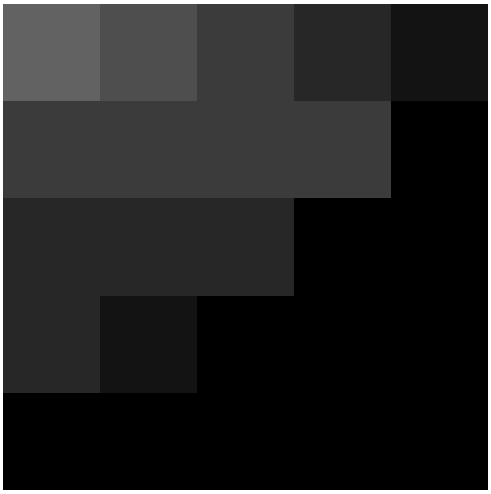


Figura 4.6: Exemplo de mapa de altura simplificado representando o terreno apresentado na Figura 4.5.

É recomendável que a quantidade de triângulos seja uma potência de dois. Ao importar um mapa de alturas a Unity escala a imagem fornecida para que haja um mapeamento exato de cada pixel para um vértice. Visando garantir que o mapa de alturas não seja distorcido durante a importação, recomenda-se que a largura da imagem esteja no formato adequado ($l = 2^n + 1$ com $n \in \{5, 6, 7, 8, 9, 10, 11\}$) antes de executar o *script*. A após sua execução a figura deve ser redimensionada para se tornar quadrada utilizado o algoritmo de redimensionamento "Nearest Neighbour".

Esta imagem é salva em formato RAW com 8 ou 16 bits de escala de cinza. É importante observar que 8 bits possibilitam apenas 256 diferentes níveis, enquanto com 16 bits são 65.536, um valor mais adequado para terrenos complexos. Portanto, 16 bits é a escala mais recomendada, mas infelizmente os monitores comerciais não são precisos o suficiente para representar tantos níveis de cinza [18, 31]. Até mesmo o Photoshop® suporta a representação de, no máximo, 10 bits de escala de cinza.

Levando em conta as restrições descritas na Seção 4.1.3 e o formato de mapa de alturas a ser utilizado, foi desenvolvido um *script* para criar mapas de altura. Esse *script* processa imagens representando o perfil topográfico do terreno, gerando uma imagem de mapa de alturas em 16 bits de escala de cinza, adequado para ser importado pela ferramenta da Unity.

Implementação

O *script* foi escrito na linguagem JavaScript que quando interpretada pelo programa Photoshop® tem acesso às principais ferramentas deste. As ferramentas do Photoshop® utilizadas pelo *script* desenvolvido são:

- Conta-gotas: ferramenta que lê informações de cor em determinada coordenada.
- Seleção retangular: seleciona uma área limitada por quatro pontos.
- Preenchimento: preenche a seleção atual com uma determinada cor.

O algoritmo para criação da nova imagem varre a imagem verticalmente procurando a primeira ocorrência de um pixel preto, que representa a superfície do terreno. Esse procedimento é feito para toda a largura da imagem e cada vez que ele é finalizado é criada uma coluna vertical de um pixel de largura cuja cor é definida pela posição vertical do primeiro pixel preto. Esse processo é representado pela Figura 4.7.

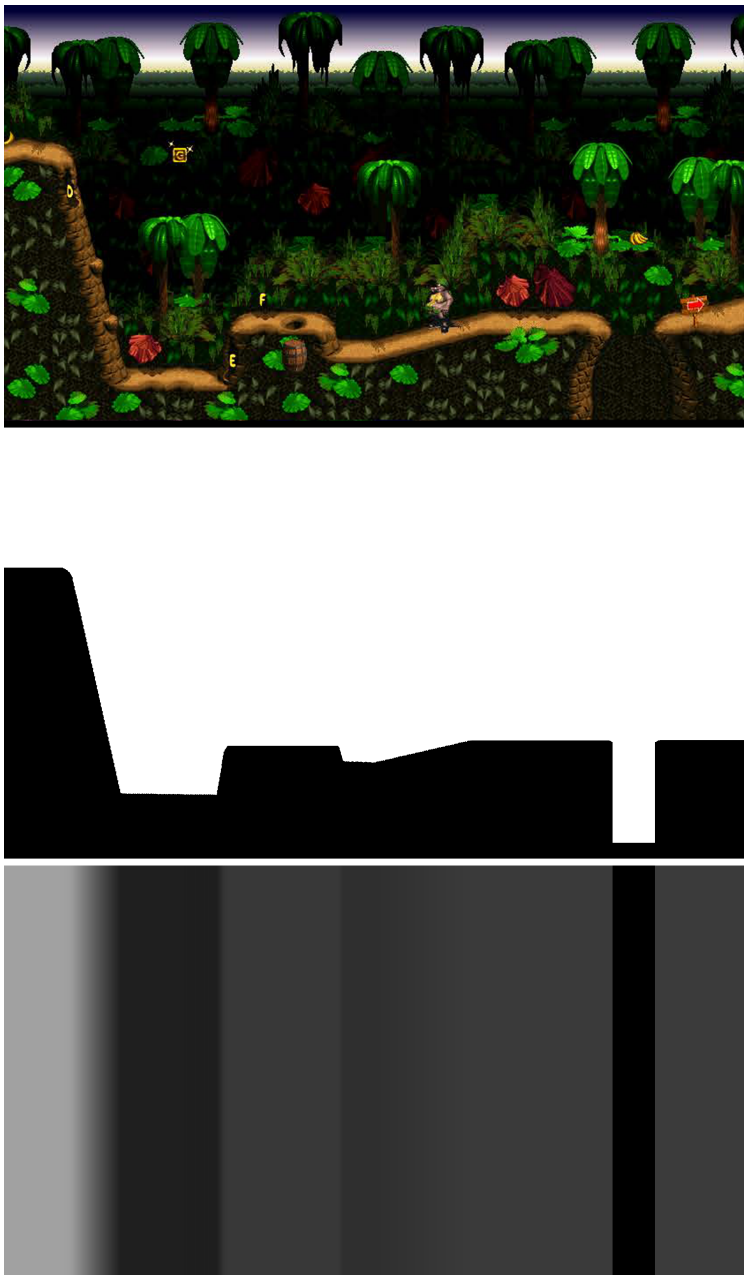


Figura 4.7: Exemplo do processamento feito pelo *script* implementado. A partir da imagem de perfil é gerado o mapa de altura.

Este algoritmo é apresentado na Listagem 4.1. Cada função apresentada possui um código específico que chama as funções da API disponibilizada pela Adobe® para interação com as ferramentas do Photoshop®.

```
function createHeightMap(){
  //Armazena as dimensoes do arquivo
  var width = activeDocument.width
  var fileHeight = activeDocument.height

  //garante que cada pixel vai ter 16 bits
  setFileFormat(BitsPerChannelType.SIXTEEN)

  //itera por todas as colunas
  for (var i=0; i<width; i++){
    //cria um retangulo que seleciona todos os pixels da coluna
    selectColumn(i)

    //Utiliza as cores dos pixels para obter a altura para aquela coluna
    var columnHeight = getColumnHeight(i)

    //usa a razao entre a altura da coluna e altura do arquivo para
       obter o tom de cinza correto para esta coluna
    var color = colorForRatio(columnHeight/fileHeight)

    //preenche toda a coluna com a cor obtida na etapa anterior
    fillWithColor(color)
  }
}
```

Listagem 4.1: Algoritmo implementado

A função `getColumnHeight()` representa a maior parte do processamento. Ela provou ser um grande gargalo no processamento da imagem. Por isso, nela foram feitas duas otimizações: busca binária e utilização de uma heurística.

Por definição, uma busca binária exige que os elementos estejam ordenados. Como neste caso o *script* procuran a coordenada y do último pixel branco essa ordem é suficientemente definida, para cada coluna, por:

- todos os pixels brancos estão acima do chão;
- todos os pixels pretos estão abaixo do chão.

Os pixels com valor absoluto maior ou igual a 32.768 são considerados brancos e os pixels com valor absoluto inferior a 32.768 são considerados pretos.

A heurística utilizada assume que o terreno tem poucas seções muito íngremes. Assim, é possível assumir que o primeiro pixel preto terá uma altura semelhante à encontrada na iteração anterior. Caso ele não seja encontrado é feita a varredura completa como no início da execução.

Pós processamento

Por ser um programa de edição gráfica, o Photoshop® possui algumas ferramentas que podem ser utilizadas para análise e pós processamento do resultado, como controle de histograma, embaçamento e redimensionamento.

Como dito anteriormente, as nuances registradas em imagens de 16 bits dificilmente são visíveis em monitores de vídeo comumente utilizados.. Para visualizar essas nuances

é possível fazer modificações na imagem baseadas no histograma (ferramenta *levels*, no Photoshop®) para analisar como as inclinações do terreno são apresentadas.

Caso seja detectado pela análise do histograma que essas inclinações estão muito bruscas, é possível utilizar a ferramenta *motion blur* para criar uma transição mais suave de uma coluna para outra. Esse efeito também pode ser controlado ao se redimensionar a imagem.

4.2.4 NGUI

No final do desenvolvimento do cenário e da simulação física do personagem foi constatado que o jogo iria precisar de alguns ajustes de otimização. Uma das otimizações aplicadas foi a utilização da biblioteca NGUI(*Next-Gen User Interface*) [7]. Esta biblioteca tem como principal característica a redução de *DrawCalls* executadas pela Unity. *DrawCalls* é chamada de renderização de cada objeto que está na tela do jogo. A NGUI consegue renderizar todos os elementos da HUD (*Head Up Display*) do jogo em uma única *DrawCall*. Para isso, a biblioteca faz uso de um atlas. O atlas é uma grande imagem composta por um arranjo das imagens de cada elemento da HUD do jogo. Para fazer o atlas a própria biblioteca disponibiliza uma ferramenta chamada `AtlasMaker` que permite selecionar as imagens que irão compor o atlas.

A biblioteca permite renderizar a HUD por meio de um painel onde devem estar todos os elementos que serão mostrados. Os elementos de um painel devem conter imagens de um só atlas. Fazendo assim, a biblioteca primeiro processa qual parte do atlas, que é uma imagem só, que será renderizada em cada parte da tela, reduzindo assim o número de *DrawCalls*. Este painel é relacionado a uma âncora (`Anchor`) que controla a posição dos elementos em relação à resolução do jogo. Esta ancora pode, por exemplo, ser configurada para a posição dos elementos se tornar relativa ao canto superior direito do jogo. Assim, independentemente da resolução do jogo, os elementos terão sempre o mesmo tamanho e estarão a uma distância fixa do canto superior direito. Além disso, as âncoras são ligadas a uma câmera independente que renderiza os elementos da HUD sobre os elementos do jogo, independentemente de sua profundidade. Toda essa hierarquia pode ser melhor visualizada na Figura 4.8.

Para o projeto foi criado somente um atlas, com todas as imagens criadas para a HUD. Essas imagens foram todas desenhadas no Photoshop® e posicionadas no atlas "FigurasHUD" como mostra a Figura 4.9. Foram criadas duas âncoras, uma posicionada relativa ao canto superior esquerdo, onde ficam as informações do jogo, como pontuação e tempo e outra, central, onde são mostradas as imagens dos desafios de abrir baús e a tela do final do jogo (Figura 4.8).

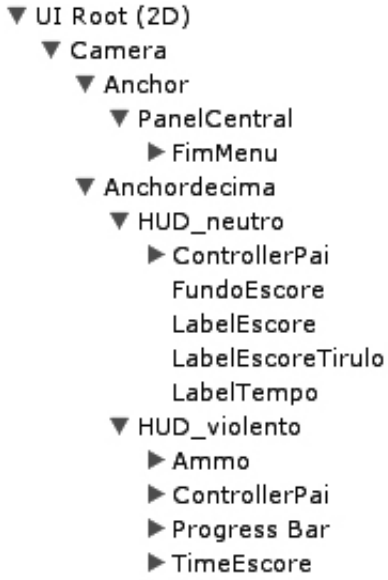


Figura 4.8: Árvore mostrando a organização dos elementos da NGUI, criados para o contexto da HUD.

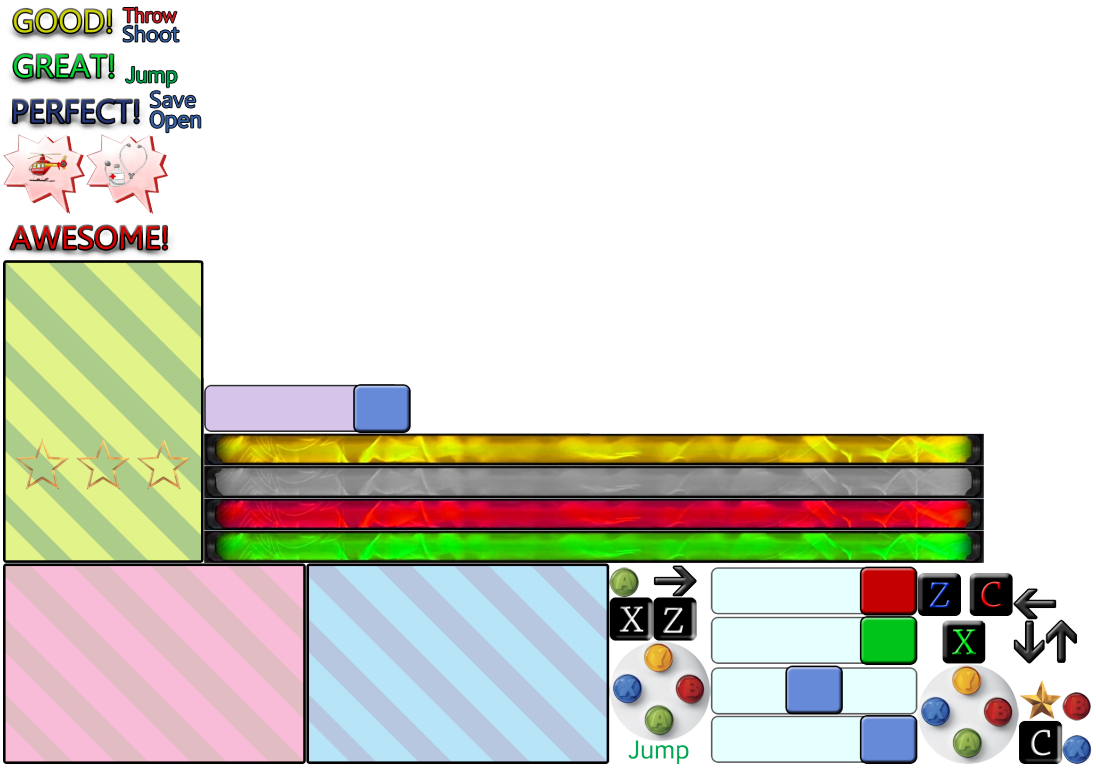


Figura 4.9: Atlas gerado pelo conjunto de imagens que compõe a HUD do jogo.

4.3 Editor

Nesta seção é explicado o funcionamento dos *scripts* desenvolvidos para serem executados diretamente no editor da Unity. Para melhor entendimento, primeiro é feita uma explicação das principais janelas da Unity.

4.3.1 Janelas da Unity

Na janela *Project* (Figura 4.10) são mostrados os *assets* do jogo que incluem: texturas, sons, modelos, cenas e *scripts*. As cenas correspondem a diferentes estágios do jogo, incluindo, assim, o menu e as fases do jogo. Os *scripts* para o jogo são codificados em C#, umas das linguagens aceitas pela Unity. Os *scripts* podem ser componentes, classe estática ou uma extensão do editor.

Cada componente desenvolvido, é na verdade uma classe e pode conter atributos serializados. Por padrão, todos os atributos públicos são serializados. Na janela *Inspector* (Figura 4.10) é possível visualizar esses atributos serializados e alterar seus valores para cada objeto separadamente. Esses atributos podem inclusive ser referências para outros *assets* do jogo, desde que esses estendam a classe `Component`. Por exemplo, a classe `MetalSlugController` tem os atributos `jumpSpeed` que define a velocidade que o personagem pula e `jumpSound` que determina o som executado ao pular. Pelo editor é possível alterar o valor do `jumpSpeed` e atribuir o som desejado ao `jumpSound`, inclusive durante o *runtime*.

Dentro da janela *Scene* (Figura 4.10), é possível visualizar uma cena do jogo por uma câmera livre, ou seja, uma câmera que permite ver todo o cenário, diferente da câmera quando se está jogando. Através desta visão é possível manusear os objetos do cenário para o lugar desejado e verificar se o tamanho, texturas e outros fatores estão corretos. Na janela *Hierarchy* (Figura 4.10) é possível ver uma lista de todos os objetos de uma cena.

Figura 4.10: Tela do editor da *Unity*. Em 1, janela *Hierachy*. Em 2, janela *Scene*. Em 3, janela *Project*. Em 4, janela *Inspector*.

4.3.2 Desenvolvimento dos modos

Normalmente, seria criada uma cena para cada modo do jogo, pois em cada modo os objetos do cenário são diferentes, exceto pelo terreno. Além disso, existem também pequenas nuances no posicionamento dos objetos no cenário, conforme as dificuldades, o que justifica fazer também uma cena por dificuldade. Como cada modo tem três dificuldades, isso iria resultar em um total 9 cenas por fase totalizando em 45 cenas diferentes para serem montadas, haja visto que o jogo tem cinco fases.

Essa quantidade de cenas iria dificultar muito o desenvolvimento e a realização de manutenção no *level design* jogo. Para contornar este problema todos os modos e dificuldades foram montados em uma única cena. Porém, isso não é simples, haja visto que em uma mesma cena irá conter todos os elementos das possíveis 9 cenas que estariam ali. Para fazer isso foi criado um componente chamado `DimensionSelector` que permite escolher em qual dificuldade e em qual modo um determinado objeto deveria existir. Assim, todos os objetos que não são comuns a todos níveis de dificuldade e a todos os modos do jogo devem ter esse componente para controlar a sua existência em um determinado momento. Por exemplo, deve-se atribuir o componente `DimensionSelector` ao objeto `MovingEnemy` que é o inimigo padrão do jogo violento. Após atribuir, o inimigo selecionado deve ser configurado de modo que esse só irá aparecer no modo violento, como era esperado, como pode ser visto na Figura 4.11.

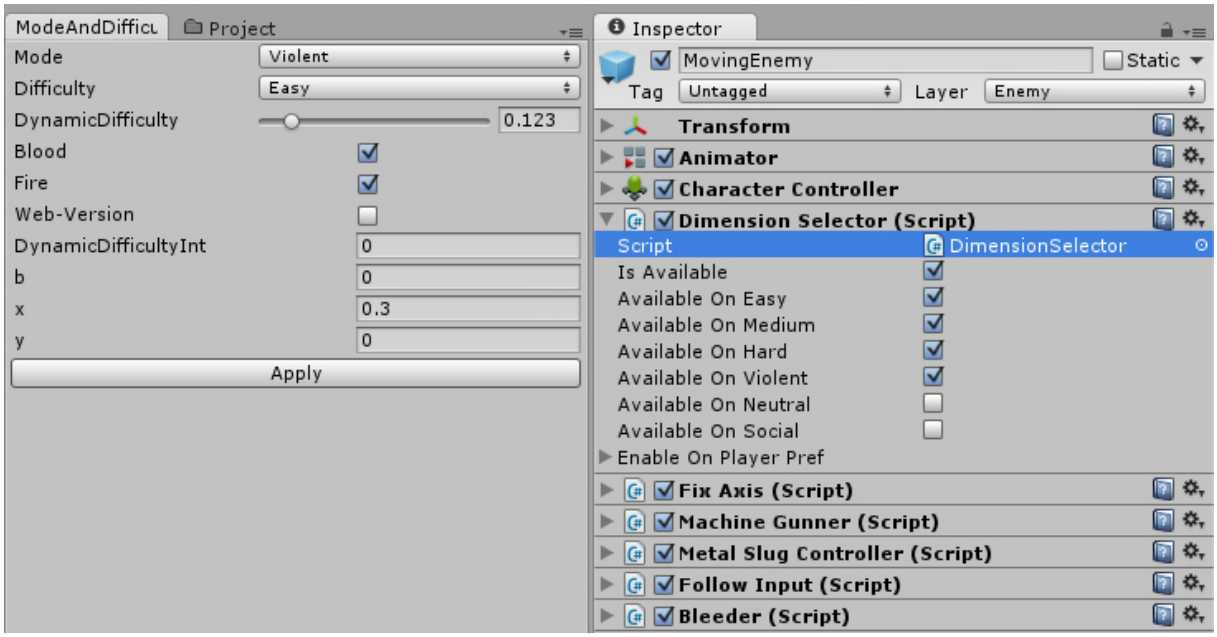


Figura 4.11: Mostra como o componente `DimensionSelector` é aplicado ao objeto `MovingEnemy`.

Este tratamento, de desaparecer com o objeto de cena, deve ocorrer não só durante o jogo, mas também no editor, para ser possível manusear os objetos da cena sem a poluição visual que iria trazer os outros objetos de outros modos e dificuldades.

4.3.3 Janela de editor

Para fazer todo esse complexo controle foram desenvolvido três *scripts*: o `ModeAndDifficulty`, o já citado `DimensionSelector` e o `DifficultyManager`. O *script* `ModeAndDifficulty` tornou possível escolher o modo e a dificuldade do jogo de dentro do próprio editor da Unity, o que anteriormente só era possível acessando o jogo e escolhendo através do menu. Para fazer isso, o *script* é colocado em uma pasta especial dentro do projeto, permitindo reconhecer que seja reconhecido como uma janela da Unity. Assim, através desta janela, é possível controlar o modo e o nível de dificuldade diretamente do editor, possibilitando manipular os elementos de uma determinada condição sem precisar trocar a cena ativa. Além disso, a janela permite modificar outras configurações do menu de opções dos modos, como por exemplo, se terá sangue ou não no jogo. Por exemplo, quando a opção "Fire" está marcada e fogo no cenário irá aparecer e ao desmarcar o fogo irá desaparecer, como pode ser visto na Figura 4.12.

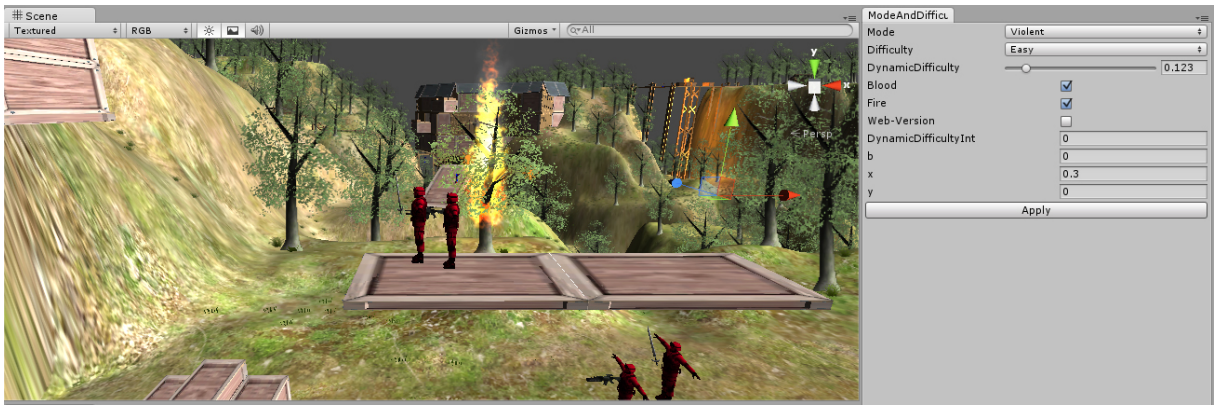


Figura 4.12: Mostra como a janela `ModeAndDifficulty` afeta a *Scene*.

O *script* `DimensionSelector` é responsável por esconder os objetos que não pertencem ao modo e o nível de dificuldade que se está trabalhando no momento. Ele deve ser atribuído a todos os objetos que variam de acordo com o modo e a dificuldade. Para cada objeto deve ser informado em qual destes modos e dificuldades o objeto deve aparecer. O *script* possui dois comportamentos diferentes: quando o jogo está em *runtime* e quando não está. Isso é possível de ser verificado através do método `isPlaying` da classe *Application*, provida pela Unity.

Vale notar que o tratamento para a dificuldade e o modo do jogo devem ser diferentes entre si. O modo não muda no decorrer da cena, mas a dificuldade, quando é configurada para ser dinâmica, pode mudificar-se no decorrer da cena. Além disso, os objetos, ao serem excluídos, devem ser tratados de maneira diferente no editor e no *runtime*, haja visto que quando um objeto é retirado no editor ele é excluído definitivamente da cena, ao contrário de quando se está em *runtime*.

Sabendo disso, o *script* apenas desabilita os componentes de renderização dos objetos que não pertencem ao modo ou dificuldade atual, quando o jogo não está em *runtime*. No exemplo retratado na Figura 4.13, a tábua sobre a ponte só deveria aparecer no nível de dificuldade fácil. Quando o nível de dificuldade é configurado para difícil a tábua some, mas continua fazendo parte da cena, como mostrado na janela *hierarchy*, onde são mostrados os objetos da cena.

Em *runtime* duas ações diferentes são realizadas. Exclusivamente para o modo o *script* destrói os objetos que não estão disponíveis no modo escolhido. Para os níveis de dificuldade, o *script* só é executado quando o objeto se torna visível ao jogador. Assim que o objeto se torna visível, o *script* verifica qual é a dificuldade atual do jogo e exclui o objeto de acordo com sua disponibilidade levando em conta a dificuldade do momento. Além disso, os objetos não excluídos são registrados na classe `DifficultyManager` que passa a controlar a dificuldade do objeto. Esse controle feito `DifficultyManager` é feito durante o *runtime* e acontece somente enquanto o objeto está visível ao jogador. Todo este controle dos objetos que variam de acordo com o modo e a dificuldade está melhor explicado na Seção 4.3.4 a seguir.

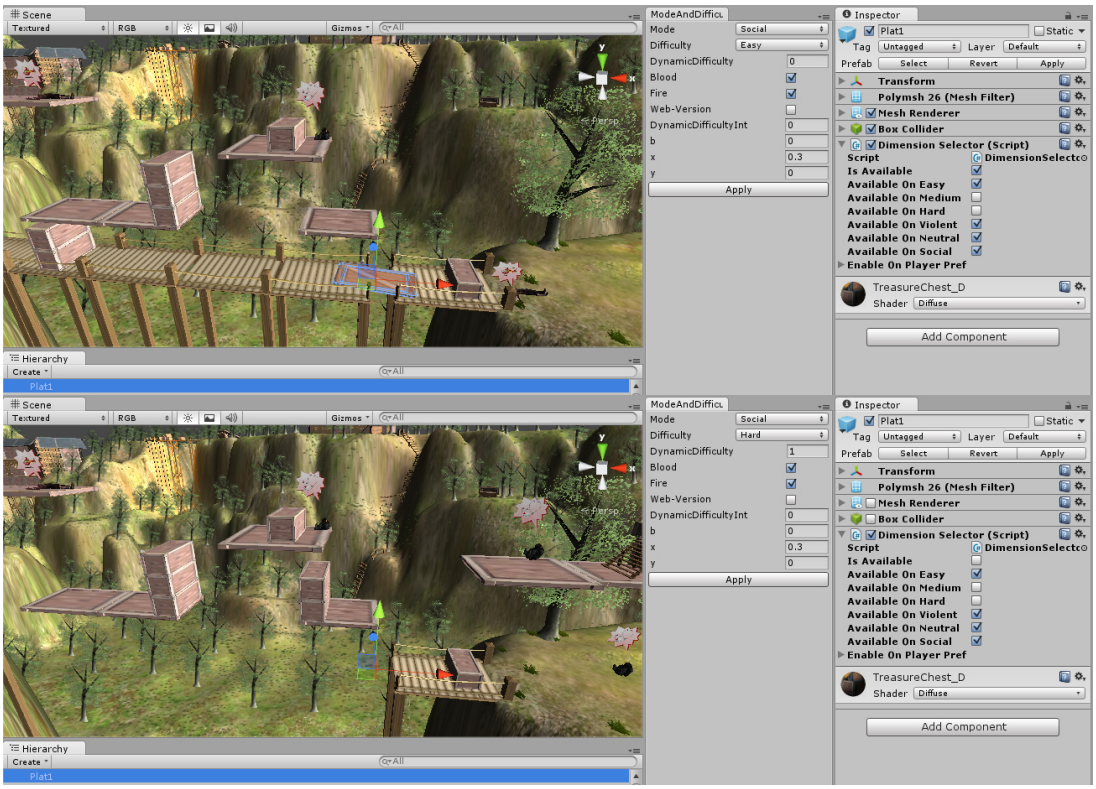


Figura 4.13: Mostrando dois casos: quando o objeto visível (em cima) e invisível (em-baixo).

4.3.4 Gerenciador de Dificuldade

A coleta de dados possui, além do propósito da pesquisa para a qual foi desenvolvida, outras funções dentro do jogo, como exemplo prover informações para o gerenciador de dificuldade. O papel dele é traduzir dados do jogo em valores absolutos que podem ser usados para controlar a dificuldade do jogo. Esses valores são coletados pela classe `Data` previamente explicada na Seção 4.4.1.

O gerenciador de dificuldade trabalha com a associação entre causa e efeito descrita anteriormente, onde as causas são dados coletados pela classe `Data` e o único efeito é a dificuldade do jogo.

Em prol do desacoplamento entre o gerenciador e o código específico do jogo, a dificuldade do jogo não é alterada diretamente. Para isso o *script* gera um valor que representa a dificuldade para uma instância da classe `Efeito`, que efetivamente modifica o comportamento do jogo. O valor passado pertence ao intervalo $[0, 1]$, onde 0 é muito fácil e 1 muito difícil.

Essa técnica de gerenciamento proposta se diferencia de uma rede neural por não haver treinamento automatizado. As funções que processam a "entrada" para gerar a "saída" do gerenciador são definidas por um *game designer*, por meio de um *script* de editor. Esse *script* é uma visualização customizada conhecida como *Custom Inspector* [36]. Ela controla a visualização de instâncias do componente gerenciador, conforme exemplificado na Figura 4.14. Por meio desse *script* são gerenciadas as causas e efeitos conforme descritos a seguir.

Causas

As causas são gerenciadas pela classe `Data` e por diversas outras classes que modificam seus dados. Como exemplo de causa podemos citar o número de vezes que o personagem morre. O gerenciador guarda apenas uma lista de `string` com todos as chaves que podem ser consideradas causas.

O *Custom Inspector* do gerenciador apresenta também *sliders* para simular os valores durante o modo de edição que pode ser utilizada para ter uma melhor visão dos efeitos.

Efeitos

Os efeitos são instâncias da classe `Effect` e são gerenciados pela classe `Difficulty-Manager`. Cada efeito pode afetar todos observadores registrados. Como, por exemplo, no caso de um efeito que controle a dificuldade de pulos e buracos, seria registrado todas as pontes ou outras estruturas que possam ser modificadas ou removidas para que alterar a dificuldade de pular sobre um buraco. Esse comportamento é melhor explicado na Seção 4.4.4 adiante.

Curvas

As curvas recebem esse nome pois seu principal atributo é uma curva que mapeia os valores de causas dos efeitos. Para os testes realizados foi implementada uma classe `DifficultyCurve` focada nas necessidades de se controlar a dificuldade do jogo. Pelo *Custom Inspector* do gerenciador é possível controlar todos os atributos dessa classe assim, como editar a curva. Como a curva é implementada por curvas Bézier, esta edição se dá pelo controle de vértices e âncoras, assim como por um programa de edição de curvas vetoriais. Essas curvas podem ser vistas na imagem 4.14.

A princípio, todas as causas possuem uma curva relacionando estas aos efeitos. Isso pode ser controlado com um atributo booleano `active` de cada curva. Esse atributo define apenas se a curva vai ser processada ou não, sem afetar os outros atributos. Como, por exemplo, a curva que relaciona numero de pontos e dificuldade deve ser crescente, pois quanto maior a pontuação do jogador, mais difícil o jogo deve ficar. Analogamente, no caso da relação do número de mortes e a dificuldade, ela deve ser decrescente.

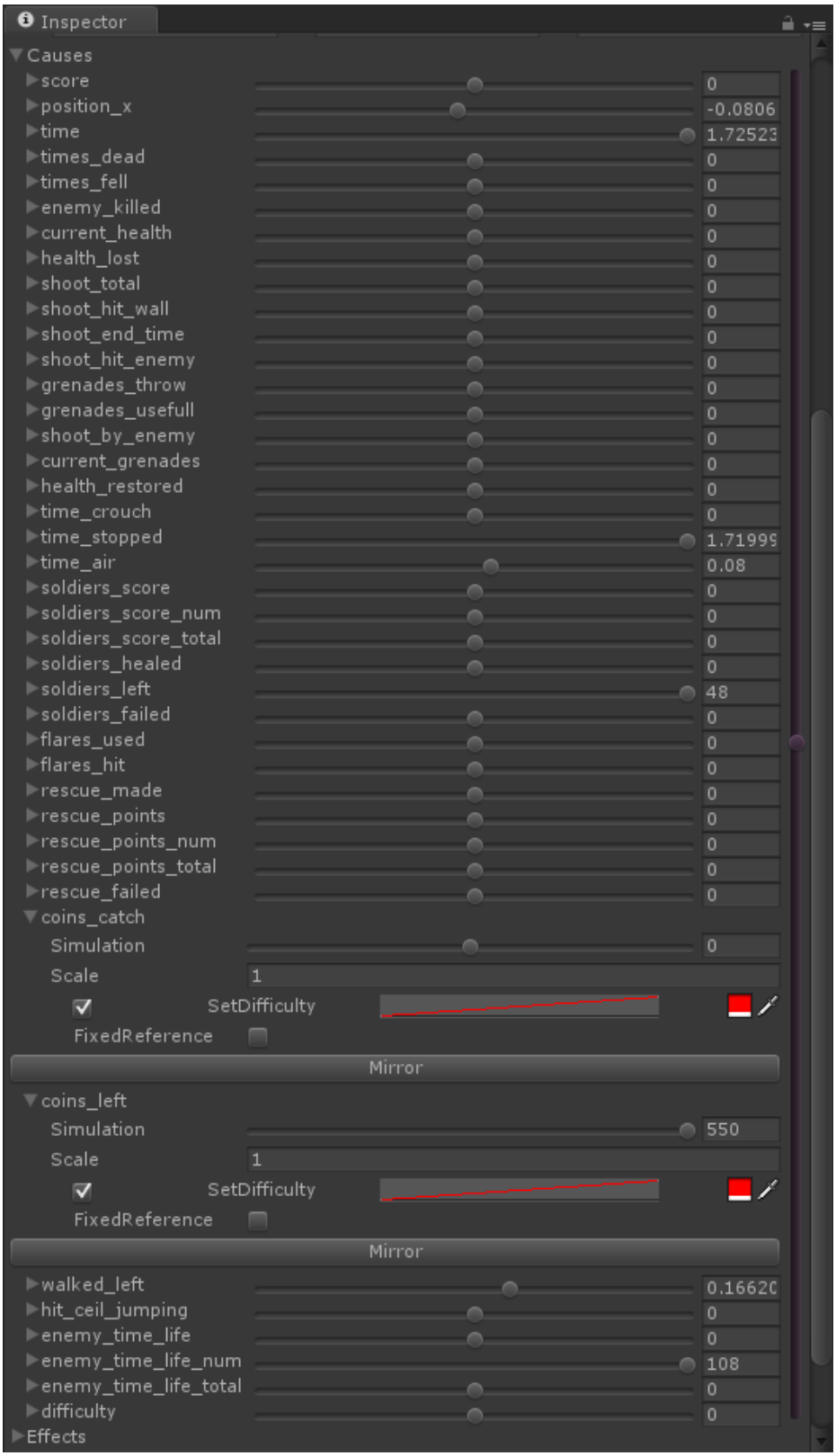


Figura 4.14: Exemplo de visualização do *Custom Inspector* da classe de gerenciamento de dificuldade.

Adaptações

Durante os testes iniciais foi detectada a necessidade de inibir certas curvas de acordo com determinadas condições. Para isso, foi implementada a opção de, para certas causas, somente considerar a causa ativa se a condição for satisfeita. Como, por exemplo, ao tratar a informação de danos causados a inimigos, esse dado só deve ser considerado caso o jogador tenha dado algum tiro ou de alguma outra maneira tenha tentado causar dano.

4.4 Runtime

Após as configurações descritas anteriormente, os *scripts* são compilados e diversos atributos são serializados para serem utilizados durante a fase de execução do jogo, denominada *runtime*. Nessa fase o comportamento de alguns *scripts* é diferente, e é nela que os dados são coletados e processados. Nesta seção são apresentados detalhes de implementação da coleta de dados além do tratamento do modo e da dificuldade durante o *runtime*.

4.4.1 Coleta de dados

A classe `Data` possui diversos atributos estáticos que podem ser acessados por qualquer classe para registrar informações coletadas pelo jogo. O código para registrar essas informações está em cada componente, tornando o código específico do jogo altamente acoplado à classe `data`. Esta decisão foi tomada em prol da performance do jogo e da quantidade de informações que se desejava armazenar.

As informações são registradas em uma lista de objetos do tipo nativo `Dictionary<string,float>`, ou seja, são definidas por uma `string` e armazenadas em um `float`. Esses valores possuem comportamentos bem característicos, isto é, alguns são sempre normalizados (pertencem ao intervalo $[0, 1]$) e outros são sempre incrementados. Estes diferentes comportamentos são tratados posteriormente pelo sistema que utiliza os dados. Todos os valores registrados estão listados na Figura 4.14.

Cada um destes dicionários representa um determinado período (vinte segundos no jogo desenvolvido), sendo criado um novo dicionário a cada ciclo. Os dicionários são armazenados em ordem pela classe `Data` para que possam ser feitas análises comparativas com o histórico.

4.4.2 Código específico

Para a execução do jogo compilado a performance é um dos mais importantes fatores. Por isso, os comportamentos de algumas das ferramentas desenvolvidas possuem códigos diferentes para a versão de editor e para a versão compilada. Esta diferenciação de código é obtida pelo uso das diretivas de compilação condicional [28] `#if`, `#else` e `#endif`.

Para entender o que é calculado em *runtime* é importante entender alguns métodos importantes da Unity apresentados a seguir. A *engine* necessariamente executa estes métodos em todos os componentes que os implementem em ocasiões específicas. O comportamento padrão é que os métodos sejam chamados apenas enquanto o jogo está sendo executado (seja no editor ou na compilação). Para sair do comportamento padrão,

atingindo assim o comportamento desejado da *engine* para este trabalho foi utilizada a metainformação `[ExecuteInEditMode]` no *script* que controla os modos e as dificuldades.

Awake

O método `Awake()` é similar ao de um construtor, sendo executado após o carregamento da cena. A principal diferença dele para um construtor é que o método `Awake()` é chamado após os atributos serializados serem populados, de modo a possibilitar o uso de informações que foram previamente serializadas por meio do editor.

Update

O método `Update()` é chamado a cada quadro do jogo, para cada componente instanciado e ativo. A taxa de quadros (ou FPS - *frames* por segundo) do jogo é facilmente influenciada por este método pois ele é chamado com muita frequência. Por outro lado se a taxa de quadros estiver baixa, este método será executado com menor frequência.

OnBecameVisible / OnBecameInvisible

Em cenários muito grandes, objetos fora da tela podem consumir muito processamento desnecessariamente. Os métodos `OnBecameVisible()` e `OnBecameInvisible()` são chamados automaticamente pela *engine* e possibilitam controle preciso em componentes que possam causar um impacto na performance mas que não precisam ser executados se o objeto não estiver sendo visto pelo jogador. É importante ressaltar que o método `OnBecameVisible()` só é chamado em objetos que possuam um componente de renderização ativo.

4.4.3 Modo

Quando uma cena é carregada, o modo já está definido e certamente não será alterado. Nesse momento já é possível ter certeza se um determinado objeto deve fazer parte do modo que foi definido no menu do jogo. Para lidar com essa escolha existem três possíveis implementações:

- Modo definido por cena: para cada modo é criada uma cena diferente. Considerando a proposta inicial de todo o *game design* ser feito em um cenário, isso demandaria duplicação e limpeza das cenas antes da compilação. Essa é uma solução que possibilita a melhor performance, pois minimiza a quantidade de objetos na cena. Por outro lado, aumenta o trabalho de desenvolvimento e cria duplicação de dados, podendo levar a inconsistências indesejadas [26].
- Análise sob demanda: todos os objetos são inicializados normalmente, sendo removidos apenas quando estritamente necessário. Essa alternativa acarretaria em todos os objetos serem completamente inicializados e potencialmente executados até o momento em que precisarem ser desativados. Isso implicaria em maior uso de memória e de tempo de CPU. Além disso, desativar os objetos durante o jogo pode causar um impacto na taxa de quadros por segundo.

- Limpeza durante a inicialização: a cena é carregada normalmente e o *script* que controla a qual modo do jogo os objetos pertencem cuida de desativar o objeto durante sua inicialização. Esse modo traz aumento no tempo de carregamento e de inicialização da cena, mas possui a mesma performance da duplicação de cena após os objetos serem destruídos - salvo nos casos ligados ao coletor de lixo.

Para a implementação do jogo, optou-se pela opção de limpeza durante a inicialização. Esta operação concentra o processamento mais pesado, mas garante maior performance quando o jogador estiver sobre o controle do personagem, que é o momento quando uma boa performance é mais necessária.

Essa implementação é feita chamando o método `TestMode()` no método `Awake()`, explicado anteriormente. Esse método é implementado no componente descrito na Seção 4.3.2. Esse método utiliza 3 atributos booleanos que são utilizados para testar se aquele objeto deve estar visível na cena atual. Este teste também cuida de remover ou esconder os objetos que não façam parte daquele modo, conforme mostrado na Listagem 4.2. Tais atributos são definidos pelo *game designer* no editor da Unity. Seus significados são:

- AvailableOnViolent: se objeto deve existir no modo **violento**;
- AvailableOnNeutral: se objeto deve existir no modo **neutro**;
- AvailableOnSocial: se objeto deve existir no modo **social**.

```
1  bool TestMode() {
2      switch (mode) {
3          case Mode.Violent:
4              if (AvailableOnViolent) {
5                  return true;
6              } else {
7                  Disable();
8                  return false;
9              }
10         case Mode.Neutral:
11             if (AvailableOnNeutral) {
12                 return true;
13             } else {
14                 Disable();
15                 return false;
16             }
17         case Mode.Social:
18             if (AvailableOnSocial) {
19                 return true;
20             } else {
21                 Disable();
22                 return false;
23             }
24         default: //Mode.All ou outro modo que a ser implementado
25             return true;
26     }
27 }
```

Listagem 4.2: Detecção de modo.

O método `Disable()`, chamado diversas vezes na Listagem 4.2, possui dois comportamentos diferentes: um no modo de edição e outro quando o jogo está em execução. O segundo acontece tanto no editor da Unity quanto no jogo compilado. A implementação desse método é mostrada na Listagem 4.3.

```
1  void Disable() {
2  #if UNITY_EDITOR
3      //Testa se o jogo esta rodando
4      if (Application.isPlaying) {
5          //Destroi o objeto enquanto o jogo esta sendo executado (acao
               revertida automaticamente ao voltar para modo de edicao)
6          Destroy(gameObject);
7      } else {
8          //Em modo de edicao, esconde o objeto sem remove-lo da cena
9          Simulate(false);
10     }
11 #else
12     //Remove o objeto da cena, evitando qualquer computacao
           desnecessaria sobre ele
13     Destroy(gameObject);
14 #endif
15 }
```

Listagem 4.3: Remoção de objetos.

O método `Disable()` é chamado em diversas situações, tanto pelo método `Update()` do próprio componente quanto por *scripts* do editor que controlam os modos e os níveis dificuldades. Nas linhas 6 e 13 da Listagem 4.3 são feitas chamadas para o método `Destroy()`. Este método e seu argumento `gameObject` são herdados da classe base de todos os componentes da Unity [35].

A chamada ao método `Simulate()` serve para garantir que o objeto estará ativo. A situação na qual os objetos podem não estar completamente ativos (ou seja, invisíveis) acontece quando, no editor, algum modo é definido para testes e portanto só objetos daquele modo devem estar visíveis.

Durante a compilação, o jogo não reativa automaticamente todos os objetos. Isso é feito, nesta etapa, pelo método `Simulate()` (linha 9 da Listagem 4.3).

Para completar a eficiência dessa implementação existe mais um detalhe a ser considerado, ou seja, garantir que o objeto seja destruído antes que seus outros componentes sejam executados desnecessariamente. Isso é obtido com o uso da ferramenta de ordenação da execução de *scripts* (Figura 4.15). Essa garantia também contribui para que dados fantasmas oriundos destes objetos não sejam coletados.

4.4.4 Dificuldade

Uma vez iniciado o jogo, a cena está populada com objetos de um único modo, mas de diferentes dificuldades. Para possibilitar a adaptação dinâmica, esses objetos devem continuar na cena, enquanto não forem percebidos pelo jogador.

Manter todas as dificuldades em cena pode causar um aumento de 200% no número de objetos. Portanto, se estes não forem tratados corretamente, pode levar a perda de performance. Com o uso do método `OnBecameVisible()` é possível minimizar a quantidade de processamento desperdiçada com objetos fora da tela.

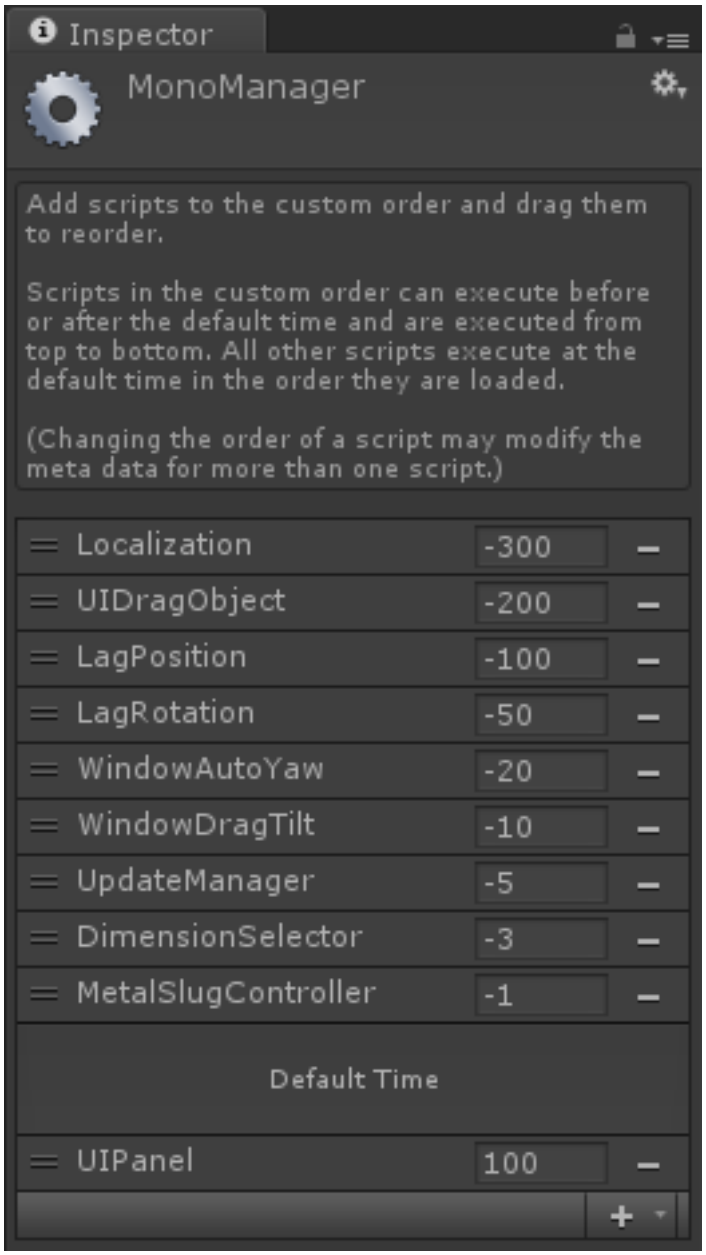


Figura 4.15: Configuração de ordem de execução de *scripts* utilizado no projeto.

Aparição

O método OnBecameVisible é utilizado para lidar com os objetos que surgem na tela, como mostra o Listagem 4.4.

```
1  void OnBecameVisible() {
2      if (TestDifficulty()){
3          //Registra este objeto como observador da dificuldade
4          DifficultyManager.Instance.RegisterObject(gameObject);
5
6          //Apos o objeto ser registrado, este componente nao tem mais
               utilidade e deve ser removido
7  #if UNITY_EDITOR
8          if (Application.isPlaying) {
9              Destroy(this);
10         }
11 #else
12         Destroy(this);
13 #endif
14     } else {
15         //Se o objeto nao pertence a dificuldade atual, ele deve ser
               destruido
16         if (Application.isPlaying){
17             Destroy(gameObject);
18         }
19     }
20 }
```

Listagem 4.4: Método para destruir os objetos que pertecem a níveis dificuldades diferentes da atual assim que os mesmos aparecem na tela.

O código na Listagem 4.4 utiliza o método `TestDifficulty()` como referência para presença do objeto na cena. Se a resposta for positiva, apenas o componente que implementa esse método (no código, `this`) é removido. Se a resposta for negativa, o objeto inteiro é destruído. A implementação do método `TestDifficulty()` é similar a do código na Listagem 4.2, ou seja, ambos utilizam atributos específicos da dificuldade e a enumeração `Difficulty`, que controla as 3 possíveis dificuldades do jogo: fácil, médio e difícil.

Na linha 4 da Listagem 4.4 é feita uma chamada para o método `RegisterObject()` do singleton `DifficultyManager`. Essa chamada faz parte da implementação do padrão Observer. Nela o sujeito de observação é o Gerenciador de Dificuldade e os observadores são quaisquer componentes ligados ao objeto em questão. Além disso, esse método adiciona o componente `DeregisterOnDestroy()` que simplesmente detecta a destruição do objeto e o remove da lista de observadores do `Difficultymanager`.

Gerenciador de dificuldade

O gerenciador pode manter a lista de observadores de três maneiras:

- Como `GameObjects`: os próprios objetos são registrados e sempre que for necessário é enviada uma mensagem para o objeto. Esta mensagem executa um método com determinada assinatura em todos os objetos que o possuem.

- Pela classe de componentes: cada efeito define exatamente quais componentes receberão seus efeitos. É mais eficiente para poucas classes, mas implica em muito acoplamento e se torna ineficiente na maioria dos casos reais nos quais existem diversos tipos de componentes que podem observar o `DifficultyManager`.
- Como interfaces: cada efeito define uma interface que deve ser implementada por todos os componentes que pretendem se registrar com ele. Assim, as atualizações do `DifficultyManager` são passadas pelo simples chamado a um único método.

No início do desenvolvimento foi utilizado o envio de mensagens para gerenciar a dificuldade. Depois de algumas análises, utilizando o *profiler* da Unity, percebeu-se a necessidade de se criar a interface `IDifficultyObserver` para melhorar a performance. Todos os componentes que implementam essa interface são listados por instâncias da classe abstrata `Effect`, sendo necessária uma implementação diferente dessa classe para cada interface desejada. Essas diferentes implementações são configuradas por meio do editor, conforme explicado anteriormente na Seção 4.3.4.

No início da execução do jogo, o gerenciador processa uma co-rotina para cada efeito configurado. A co-rotina é executada na mesma *thread* que todo o resto de jogo, mas pode ser suspensa, simulando uma execução paralela para todos os diferentes efeitos. Da mesma forma, a co-rotina processa constantemente todas as causas, calculando o desvio padrão de seus valores de acordo com as configurações especificadas por meio do editor. Esse desvio padrão é comparado e aplicado à curva específica daquela causa, resultando em um valor que é passado para o efeito. Este por sua vez chama os métodos correspondentes da interface que ele gerencia nos devidos componentes dos objetos que estão atualmente registrados como observadores.

Capítulo 5

Testes

Com o objetivo de testar o jogo desenvolvido foi conduzida uma pesquisa online. Esta pesquisa teve como foco investigar se os jogos violento, pró-social e neutro estão passando para o jogador a ideia de violência, pró-socialidade ou nenhuma das duas respectivamente. Este teste é fundamental para o projeto, pois trazer a experiência adequada ao jogador é o principal objetivo do jogo desenvolvido. Além disso, o teste também verificou a dificuldade dinâmica. Foi mensurado se as dificuldades dinâmicas "sempre difícil" e "sempre fácil" estão, de fato, produzindo o efeito proposto na experiência do jogador.

5.1 Versão Online

Para adequar o jogo ao ambiente online e aos objetivos do teste foi compilada uma versão com as seguintes configurações:

- Resolução fixa de (600 X 960) pixels.
- Somente as duas primeiras fases do jogo foram incluídas, retirando a parte de tutorial. O tempo máximo para o jogador concluir estas duas fases é de 8 minutos.
- As opções que remetem à ideia de violência, como fogo no cenário e sangue, foram ativadas apenas no modo violento.
- Condição de jogo gerada é aleatoriamente.
- Identificação do jogador é obtida a partir do sistema web da pesquisa.

Com essas configurações não se fez necessário incluir no teste o menu inicial, que permite acesso às configurações, o menu de escolha dos modos e o campo ID do menu principal.

Jogos feitos em Unity podem ser executados em navegadores *Web*. Basta mudar a plataforma para "*Web Player*" e compilar normalmente. Para executar o jogo, o jogador deve instalar de um *plugin* da Unity que é compatível com os principais navegadores para Windows e MAC OS X. Não é possível executar este jogo em dispositivos móveis. O binário gerado pela Unity foi hospedado em uma conta no Dropbox™ [19], um sistema que permite arquivar e compartilhar arquivos.

5.2 EFS *Survey*

A pesquisa foi projetada juntamente com o professor Mauricio Miranda Sarmet e o grupo Pró-Games, utilizando um sistema online privado chamado EFS Survey [9]. Este é o mesmo sistema usado pelo grupo para realizar suas pesquisas e foi escolhido por já ter sido utilizado com sucesso em outras de suas pesquisas. O sistema possui recursos que facilitam a criação do questionário e permite exportar arquivos em formato `.csv` contendo os dados obtidos.

O questionário é divido em páginas. Na primeira página, é apresentada uma introdução para o voluntário, mostrando como será o procedimento da pesquisa e alertando da necessidade de se instalar o *plugin* da Unity.

Na segunda página, o participante tem acesso ao jogo. Nesta hora, o sistema armazena em qual o modo e em qual nível dificuldade o voluntário está jogando. O participante não sabe que existem outros modos de jogo e níveis de dificuldade. Além disso, ao final do jogo são armazenados todos os dados contidos na classe `Data` e três informações extras: a dificuldade em que se encerrou o jogo, o tempo que demorou a primeira fase e o tempo que demorou a segunda fase. Tudo isso é armazenado separadamente pelo sistema (EFS) para permitir uma análise posterior.

Na terceira página é onde se iniciam as perguntas. Nela, o participante expressa sua opinião sobre o jogo através das perguntas. Vale ressaltar que as perguntas do questionário neste ponto são fundamentais, pois elas são usadas para mensurar o efeito provocado pelo jogo no indivíduo. As perguntas variam na escala de 0 a 10, totalizando 11 possíveis respostas. As perguntas desta página são:

- Qual foi o nível de dificuldade do jogo?
- Quão agradável foi o jogo?
- Quão frustrante foi o jogo?
- O quanto o jogo foi excitante para você?
- O quão violento você considera o jogo que experimentou?
- O quão pró-social é este jogo?

Nas páginas quatro e cinco é traçado o perfil demográfico do participante. Como explicado no primeiro capítulo, vivências anteriores afetam o julgamento de cada indivíduo. Portanto, pode-se teorizar uma justificativa para determinada resposta com base nas informações pessoais. Por exemplo, é possível que jogadores de mais idade e com menos experiência em jogos eletrônicos tenham a tendência de achar o jogo mais violento. A quinta e ultima página é opcional e o participante somente é direcionado a ela caso este responda "sim" para a pergunta "Você tem o hábito de jogar jogos eletrônicos?" na página quatro.

As perguntas da página quatro são:

- Idade:
- Sexo:
- Nível de escolaridade:

- Formação:
- Você tem o hábito de jogar jogos eletrônicos?

As perguntas da página cinco são:

- Quantas horas você joga por semana, aproximadamente?
- Em qual horário você joga mais frequentemente?
- Quais consoles utiliza para jogar:
- Cite três jogos mais jogados por você no ultimo mês:

Para participar da pesquisa basta seguir o link gerado pelo sistema (http://ww3.unipark.de/uc/GEPS/3d2c/). Esta pesquisa ficou disponível por um período de duas semanas (de 15/06/13 a 05/07/13) e foi divulgada em diversos meios de comunicação online.

5.2.1 Comunicação JavaScript e C#

Para fazer a interação do jogo com o sistema *EFS Survey* foram utilizadas funções de comunicação entre o código JavaScript (JS) da página web e o C# utilizada dentro da Unity. Dados passados da Unity para o navegador são então repassados para o sistema EFS, como se fossem perguntas respondidas pelo participante. Isto é possível pois o sistema permite a criação de perguntas definidas pelo pesquisador. Foi criado então uma pergunta para cada dado coletado pelo jogo, como por exemplo, qual foi a dificuldade em que o usuário jogou. Essas perguntas são invisíveis, pois nem mesmo o usuário tem acesso às respostas, e portanto são respondidas for funções presentes no código JS da página de pesquisa que são chamadas pela Unity. Portanto, é na página dois que o sistema armazena o modo jogado e o nível de dificuldade juntamente com as outras dezenas de informações registradas pelo jogo.

Para melhor se adequar ao cenário web foram acrescentadas algumas funcionalidades. O primeiro problema encontrado foi o fato dos participantes, em sua maioria, não lerem as instruções dentro da janela do jogo. Note que, pelo fato do jogo decidir, a priori, de maneira aleatória, o modo a ser jogado pelo voluntário, não era possível mostrar as instruções antes do jogo ser carregado, pois cada modo tem instruções diferentes, como mostra a Figura 5.1. Para solucionar este problema, foi adicionado ao código da segunda página uma função para sortear o modo de jogo. Dessa maneira as instruções aparecem antes do jogo ser carregado e permite ao jogador ler as instruções com antecedência.

Essa mudança fez com que a cada novo carregamento da página web, fosse sorteado um novo modo. Contudo, não é interessante para a pesquisa o fato do participante poder atualizar o navegador e com isso jogar um outro modo. Isso não aconteceu quando o modo era definido pela Unity, pois esta armazena localmente os dados do jogo. Esse problema acontecia principalmente quando o voluntário necessitava recarregar a página para instalar o *plugin* necessário. Para resolver este novo desafio, é armazenado no navegador do usuário a informação do primeiro modo sorteado para não ocorrer um novo sorteio do modo ao atualizar a página. Toda essa sequencia de eventos descrita pode ser melhor visualizada na Figura 5.2. Além disso, é possível observar como foi feita a implementação em JavaScript e C# para solucionar este problema nas Listagens 5.1 e 5.2.

Figura 5.1: Em sentido horário: instruções dos jogos violento, neutro pró-social.

```
1  //Inicio
2      if( (var mode = LerCookie("Modo")) == null){ //Verifica se o cookie
           ja esta guardado
3        mode = Math.floor((Math.random()*3)); //Gera modo
4        GuardaCookie("Modo", mode); //Guarda modo em cookie
5      }
6      switch(mode){
7        case 0:
8          document.getElementById('mode0').style.display='block'; //Mostra
               instrucoes do modo 0 (violento)
9          break;
10       case 1:
11         document.getElementById('mode1').style.display='block'; //Mostra
               instrucoes do modo 1 (pro-social)
12         break;
13       case 2:
14         document.getElementById('mode2').style.display='block'; //Mostra
               instrucoes do modo 2 (neutro)
15         break;
16     }
17
18     function QualModo(){
19       u.getUnity().SendMessage("LabelRecebe", "SetMode", mode); //Envia
             modo escolhido para a Unity
20     }
21     function CloseGame(arg){
22       SalvaDados(arg); // Salva os dados do jogo
23       document.getElementById('divGame').style.display='none'; //Fecha
             jogo
24       document.getElementById('divNext').style.display='block'; //
             Mostra botao para avancar para proxima pagina
25     }
```

Listagem 5.1: Trecho de Código do JavaScript

```
1    void InicioDoJogo(){ //chamado no inicio do jogo
2      //Chama funcao "QualModo" do JavaScript
3      Application.ExternalCall("QualModo");
4    }
5
6    void SetMode(int WebMode){
7      PlayerPrefs.SetInt("Mode", WebMode); //Define dificuldade do jogo
8    }
9
10   void FimdoJogo () { //chamado ao fim do jogo
11     //Chama funcao CloseGame do JavaScript enviando dados do jogo
12     Application.ExternalCall( "CloseGame", Data.SumToArray() );
13   }
```

Listagem 5.2: Trecho de código em C# dentro da Unity

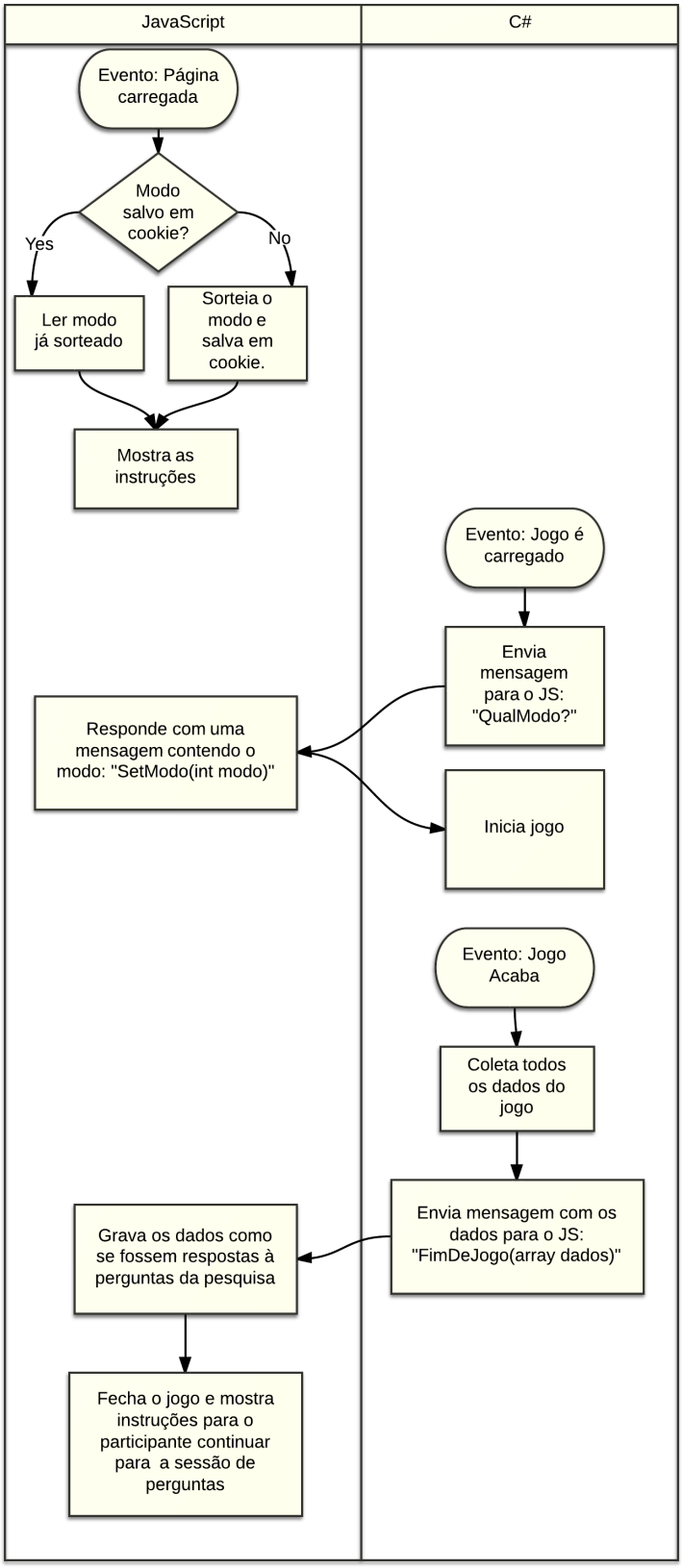


Figura 5.2: Esquema demonstrando o procedimento ao abrir a segunda página da pesquisa.

5.3 Resultados

Ao final da pesquisa os dados foram exportados para um arquivo `.csv` e analisados utilizando o programa SPSS Statistics Professional da IBM® [20]. O grupo de pesquisa utiliza uma versão proprietária dessa ferramenta na análise de dados coletados em outras pesquisas por eles aplicadas. A ferramenta foi escolhida por prover uma resposta imediata e de maneira clara às análises requeridas.

O SPSS faz uso dos comprovados métodos estatísticos Bonferroni e Games-Howell que calculam o nível de significância. Estes cálculos não são tratados neste trabalho, podendo ser consultado o livro de Field [20] para mais detalhes. Este nível de significância é importante pois mesmo quando existe uma diferença nas respostas dos participantes ao mudar uma variável, como o modo de jogo, não é possível afirmar com certeza que essa diferença foi devido à mudança da variável. O nível de significância é um valor calculado pelos métodos estatísticos citados e serve para dizer se essa diferença nas respostas foi devido à mudança de uma determinada variável, ou não. Quanto menor a significância, maior a probabilidade da variável em questão ter sido a única causa desta mudança no valor das respostas. O limiar considerado para avaliar significância é 0,05.

Utilizando o SPSS foram feitas duas grandes análises. Primeiro foi verificado se as perguntas "O quão violento foi o jogo?" e "O quão pró-social foi o jogo?" tiveram os resultados esperados ao mudar o modo. Segundo foi verificado se a pergunta "Qual foi o nível de dificuldade do jogo?" teve resultado esperado ao mudar o nível de dificuldade.

5.3.1 Análise do modo

As respostas das perguntas "O quão violento foi o jogo?" e "O quão pró-social foi o jogo?" se tornaram as variáveis desta análise. Elas variam em uma escala de 1 a 11. A variável `mode`, é o modo de jogo, podendo ser "violento", "neutro" ou "pró-social". Os resultados esperados da análise são apresentados na Tabela 5.1.

	"O quão pró-social foi o jogo?"	"O quão violento foi o jogo?"
Violento	baixo	alto
Neutro	baixo	baixo
Pró-social	alto	baixo

Tabela 5.1: Resultados esperados das respostas para as perguntas "O quão pró-social foi o jogo?" e "O quão violento foi o jogo?"

Foram realizadas ANOVAs comparando as avaliações dos níveis de violência e pró-socialidade feitas pelos participantes de cada grupo. Os resultados obtidos foram satisfatórios. Observou-se uma diferença significativa entre os grupos que jogaram o modo violento, pró-social ou neutro na avaliação do nível de violência do jogo, $F\ (2,\ 91) = 29{,}14$, $p < 0{,}001$, $\omega^2 = 0{,}38$. Análises *post hoc* identificaram que a diferença na avaliação dos participantes segue o esperado, com o modo violento avaliado como mais violento que os dois outros modos (sem diferença significativa entre os modos neutro e pró-social). Este é o resultado ideal, pois mostra que foi o fato dos usuários terem jogado nesses diferentes modos que afetou suas percepções do jogo. As médias para cada pergunta são apresentadas na Tabela 5.2.

	"O quão pró-social foi o jogo?"	"O quão violento foi o jogo?"
Violento	2,06	5,53
Neutro	1,72	1,07
Pró-social	8,41	2,24

Tabela 5.2: Médias das respostas para as perguntas "O quão pró-social foi o jogo?" e "O quão violento foi o jogo?"

A mesma análise foi realizada considerando a percepção de pró-socialidade por parte dos respondentes, e os resultados sugerem que há uma diferença significativa entre os grupos, $F\ (2,\ 91) = 83{,}50$, $p < 0{,}001$, $\omega^2 = 0{,}64$. Os testes *post hoc* mostram de que o grupo que exposto ao modo pró-social avaliou o jogo como mais pró-social do que os outros modos (e sem diferença significativa entre os modos violento e neutro).

Ambos os resultados obtidos confirmam as expectativas.

Os resultados completos obtidos pela análise dos dados utilizando os métodos Bonferroni e Games-Howell são apresentados na Figura 5.3 e os valores médios obtidos nas perguntas podem ser melhor visualizados nas Figuras 5.4 e 5.5 .

Multiple Comparisons

Dependent Variable		(I) mode	(J) mode	Mean Difference (I-J)	Std. Error	Sig.
Nada violento - Extremamente violento	Bonferroni	Violento	Neutro	4,459*	,611	,000
			Pró-social	3,286*	,611	,000
		Neutro	Violento	-4,459*	,611	,000
			Pró-social	-1,172	,643	,215
		Pró-social	Violento	-3,286*	,611	,000
			Neutro	1,172	,643	,215
	Games-Howell	Violento	Neutro	4,459*	,584	,000
			Pró-social	3,286*	,695	,000
		Neutro	Violento	-4,459*	,584	,000
			Pró-social	-1,172*	,383	,013
		Pró-social	Violento	-3,286*	,695	,000
			Neutro	1,172*	,383	,013
Nada prosocial - Extremamente prosocial	Bonferroni	Violento	Neutro	,331	,539	1,000
			Pró-social	-6,082*	,539	,000
		Neutro	Violento	-,331	,539	1,000
			Pró-social	-6,414*	,568	,000
		Pró-social	Violento	6,082*	,539	,000
			Neutro	6,414*	,568	,000
	Games-Howell	Violento	Neutro	,331	,471	,763
			Pró-social	-6,082*	,579	,000
		Neutro	Violento	-,331	,471	,763
			Pró-social	-6,414*	,639	,000
		Pró-social	Violento	6,082*	,579	,000
			Neutro	6,414*	,639	,000

Figura 5.3: Tabela comparativa entre os modos e as respostas dos participantes.

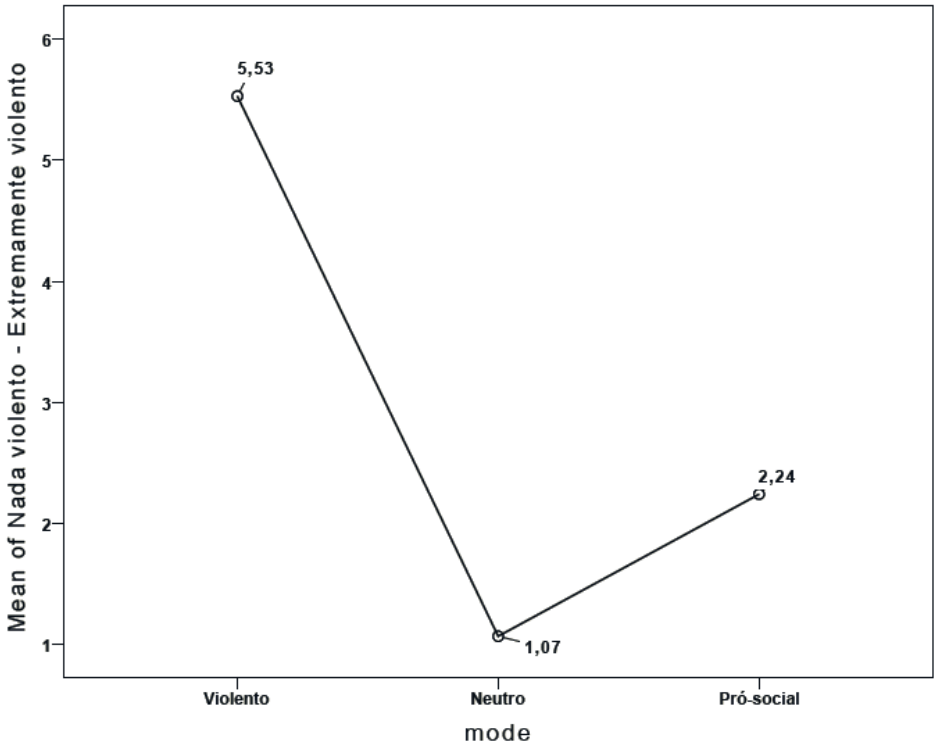


Figura 5.4: Valor médio, para cada modo, obtido na pergunta "O quão violento foi o jogo"

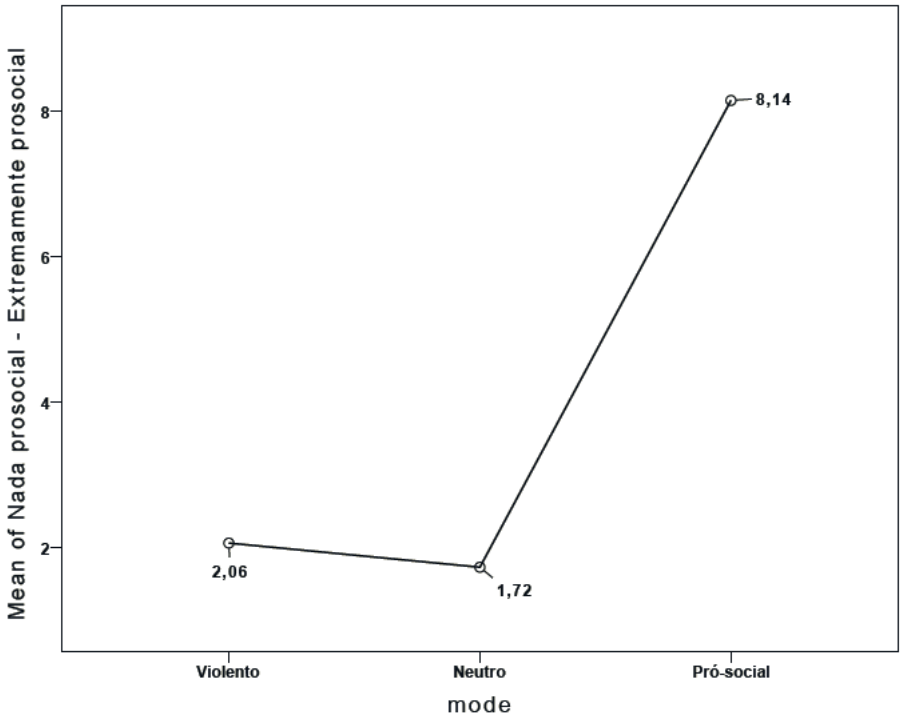


Figura 5.5: Valor médio, para cada modo, obtido na pergunta "O quão Pró-social foi o jogo"

5.3.2 Análise da dificuldade

Uma análise semelhante foi feita em relação à dificuldade. A resposta para a pergunta "Qual foi o nível de dificuldade do jogo?" se tornou a variável de análise. Ela varia em uma escala de 1 a 11. A variável `difficulty`, é a dificuldade do jogo, podendo ser "fácil", "médio" ou "difícil". Vale ressaltar que a dificuldade "médio" é estática, enquanto as dificuldades "fácil" e "difícil" são dinâmicas e configuradas para serem "sempre fácil" e "sempre difícil", respectivamente. Além disso, ambas iniciam com a dificuldade base "médio".

Os resultados obtidos, fazendo a ANOVA, foram parcialmente satisfatórios. As médias para a pergunta "Qual foi o nível de dificuldade do jogo?" são mostradas na Tabela 5.3. O resultado $F\ (2,\ 91) = 7{,}46$, $p = 0{,}001$, $\omega^2 = 0{,}12$ mostra que a dificuldade dinâmica está realmente fazendo efeito na experiência do jogador.

Dificuldade	Média
Fácil	2,42
Médio	4,54
Difícil	5,03

Tabela 5.3: Média das repostas para "Qual foi o nível de dificuldade do jogo?"

Porém, na comparação ente o jogo no nível de dificuldade "médio" e "difícil" é notado que a dificuldade dinâmica quando configurada para "sempre difícil" não está deixando o jogo suficientemente mais difícil pois não existe uma diferença significativa entre as dificuldades.

Os resultados completos obtidos pela análise dos dados utilizando os métodos Bonferroni e Games-Howell são apresentados na Figura 5.6 e os valores médios obtidos na pergunta "Qual foi o nível de dificuldade do jogo?" podem ser visualizados na Figura 5.7.

Dependent Variable: Extremamente fácil - Extremamente difícil

	(I) difficulty	(J) difficulty	Mean Difference (I–J)	Std. Error	Sig.
Bonferroni	Médio	Fácil	2,122*	,679	,007
		Difícil	-,494	,629	1,000
	Fácil	Médio	-2,122*	,679	,007
		Difícil	-2,616*	,711	,001
	Difícil	Médio	,494	,629	1,000
		Fácil	2,616*	,711	,001
Games-Howell	Médio	Fácil	2,122*	,579	,002
		Difícil	-,494	,686	,753
	Fácil	Médio	-2,122*	,579	,002
		Difícil	-2,616*	,640	,000
	Difícil	Médio	,494	,686	,753
		Fácil	2,616*	,640	,000

Figura 5.6: Tabela comparativa entre os dificuldades do jogo e as respostas dos participantes.

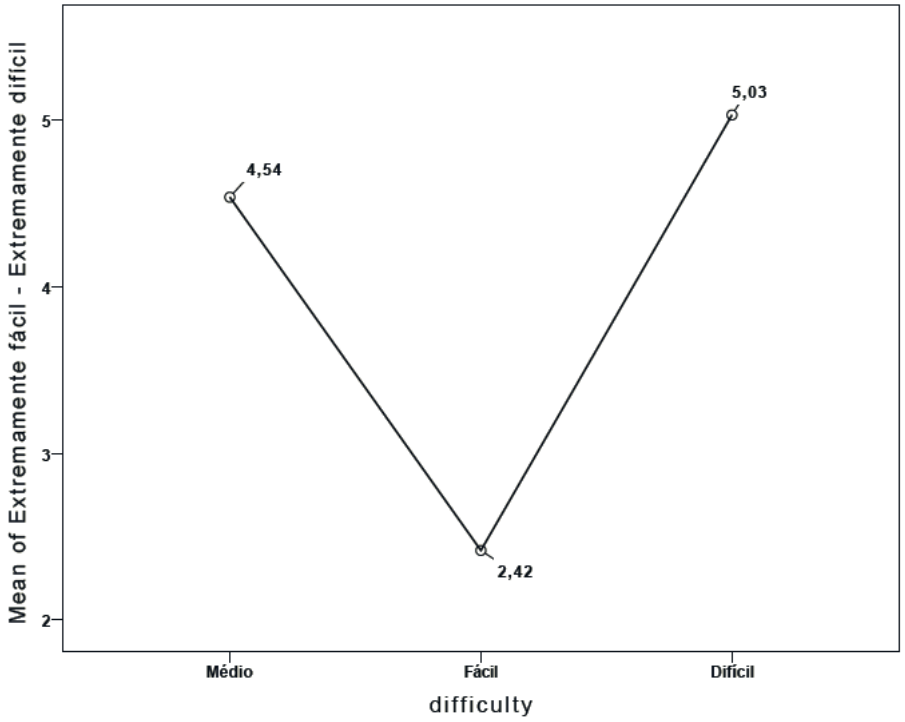


Figura 5.7: Valor médio obtido na pergunta "Qual foi o nível de dificuldade do jogo?" separado por níveis de dificuldade.

5.3.3 Outras análises

Separando os participantes em relação aos modos de jogo, foi realizada um análise em relação a percepção dos níveis de agradabilidade, excitação e frustração em função da experiência do jogo. Essa análise foi importante pois um dos requisitos do jogo é de que este não tenha diferença entre os modos, além das variáveis manipuladas. As ANOVAs não indicaram diferenças significativas, ou seja, os modos conseguiram transmitir uma experiência parecida para os jogadores, o que é um resultado positivo.

Foi realizada também uma análise do nível de dificuldade percebido em cada modo de jogo. Para isso foi analisado cada modo de jogo e cada nível de dificuldade isoladamente. O esperado para está análise é que todos os modos apresentem um nível de dificuldade semelhante, para o nível de dificuldade não seja uma variável indesejada.

No nível de dificuldade médio, a ANOVA não apresentou diferenças significativas na percepção dos jogadores. No caso do nível fácil de dificuldade, foi encontrada diferença significativa do nível de dificuldade percebido em função do modo de jogo, $F\ (2, 21) = 5{,}43$, $p = 0{,}012$, $\omega^2 = 0{,}27$. Esta diferença reside entre o modo pró-social e os demais. No caso do nível de dificuldade difícil a ANOVA também mostrou diferença significativa $F\ (2, 28) = 3{,}54$, $p = 0{,}043$, $\omega^2 = 0{,}14$. Esta diferença também reside entre o modo pró-social e os demais. Em geral, o modo pró-social foi considerado mais difícil do que os outros dois modos, mostrando que o jogo necessita de ajustes neste modo.

Capítulo 6

Conclusão

Neste trabalho foi apresentado o desenvolvimento de um jogo eletrônico para fins de pesquisa na área da Psicologia Social.

O trabalho foi desenvolvido em parceira com o grupo de pesquisa Pró-Games, coordenado pelo professor Mauricio Sarmet. Desta forma foi possível criar um jogo que atendeu às necessidades do grupo de forma customizada e ao mesmo tempo com algum grau de generalidade para que o mesmo possa ser útil em várias pesquisas futuras conduzidas pelo grupo. Foram feitas diversas reuniões conjuntas para identificação das demandas que os pesquisadores tinham que normalmente não são supridas pelos jogos comerciais utilizados atualmente.

Planejar três jogos distintos com jogabilidade similar foi um desafio. Cada elemento do *game design* foi cuidadosamente analisado. Desenvolver o jogo neutro e pró-social, sem nenhuma característica de violência, exigiu muita criatividade. São poucos os jogos de plataforma que não contêm nenhum elemento de violência, haja visto que é da natureza deste tipo de jogo o jogador evitar perigos para alcançar o objetivo. Desenvolver os três modos do jogo na mesma cena demandou planejamento e customização do editor da *engine*. Isso possibilitou a criação de cenários com quantidade adequada de obstáculos de acordo com o modo de jogo. Ao final, os elementos projetados e o terreno criado se adequaram bem aos três modos do jogo, resultando em uma experiência consistente e agradável.

Para desenvolver o jogo foi utilizada a *engine* Unity. Esta foi escolhida tendo tendo em vista a facilidade no desenvolvimento de jogos e familiarização da equipe com as suas funcionalidades. Diversos desafios foram encontrados durante o desenvolvimento. O *design* do jogo passou por várias mudanças antes de chegar à sua versão *beta*. O grupo de pesquisa ficou muito satisfeito com evolução do jogo, a arte se tornou mais atrativa e os modos mais divertidos. Para alguns problemas pontuais foram encontradas soluções inovadoras. O *script* para transformar a imagem do mapa em terreno da Unity, a maneira como foram desenvolvidos os três modos junto dos três níveis dificuldades em somente uma única cena e a ferramenta completa de gerência da dificuldade dinâmica são soluções pensadas e desenvolvidas que merecem destaque.

Ao final foi realizado um teste online com o objetivo de medir a percepção dos jogadores em relação ao jogo. Os resultados foram analisados utilizando métodos estatísticos conhecidos, se mostrando positivos e confirmando que o jogo atingiu as condições desejadas para que sejam feitas pesquisas com ele. Foi verificada a necessidade de alguns

ajustes que serão realizados antes da entrega de uma versão final do jogo. Este, como todo *software*, precisa estar em constante ajuste para se adaptar às novas exigências, havendo o comprometimento de realização de melhorias.

Como sugestão para trabalhos futuros é proposta uma maneira de minimizar a demora que existe no jogo ao começar uma cena, que deverá ser realizada sem comprometer o processo de *game design*. Para isso, deve ser criado um *script* que divida as cenas do jogo em diversas cenas específicas para cada modo no momento da compilação do jogo [40]. Além disso, é necessário que o código que carrega as cenas do jogo seja capaz de decidir quais dessas novas cenas deverão ser carregadas.

Em relação à própria pesquisa podem ser feitas extensões ao sistema para possibilitar diferentes focos ou melhorar a qualidade dos dados obtidos. Uma sugestão é criar mais opções no menu dos modos como, por exemplo, inimigos humanos ou robôs, música violenta ou passiva e visual *cartoon* ou realístico. Também seria interessante criar um servidor geral para distribuir igualmente as condições jogadas. Isso evitaria discrepâncias muito grandes na quantidade de respostas de cada condição sem interferir com a aleatoriedade requerida pela pesquisa.

Referências

[1] Adaptations NG P4. URL http://www.neogeokult.com/articles/adaptations-ng-p4/, acessado em: 13/07/13. vi, 20, 35

[2] DotA AllStars, 2013. URL http://www.getdota.com/, acessado em: 01/07/2013. 16

[3] Fat Princess, 2013. URL http://www.ign.com/games/fat-princess/ps3-14266745, acessado em: 30/01/2013. vi, 13

[4] Games with a purpose, 2013. URL http://www.gwap.com/, acessado em: 10/04/2013. 18

[5] Fat Princess, 2013. URL http://www.gepsocial.com.br/geps, acessado em: 01/02/2013. 1, 4, 6

[6] International Superstar Soccer, 2013. URL http://www.vgchartz.com/game/4721/international-superstar-soccer/, acessado em: 01/07/2013. 15

[7] NGUI, 2013. URL http://www.tasharen.com/?page_id=140, acessado em: 30/05/2013. 42

[8] Ronaldinho Soccer 98, 2013. URL http://topgameretro.blogspot.com.br/2012/06/snes-ronaldinho-campeonato-brasileiro.html, acessado em: 01/07/2013. vi, 15

[9] QuestBack AG. EFS Survey, 2013. URL http://www.unipark.com/63-1-efs-survey.htm, acessado em: 20/06/2013. 58

[10] Autodesk Inc. Autodesk FBX, 2013. URL http://www.autodesk.com/products/fbx/overview, acessado em: 25/06/2013. 36

[11] Sander Bakkes, Chek Tien Tan, and Yusuf Pisan. Personalised Gaming: A Motivation and Overview of Literature. 2012. 16

[12] Mary E. Ballard and J. Rose Wiest. Mortal kombat (tm): The effects of violent videogame play on males' hostility andcardiovascular responding. *Journal of Applied Social Psychology*, 1996. vi, 6, 8, 9

[13] Christopher P. Barlett, Craig A. Anderson, and Edward L. Swing. Video game effects – confirmed, suspected, and speculative: A review of the evidence. *Journal of Experimental Social Psychology*, 2009. vi, 1, 2, 4, 11

[14] Brock Bastian, Jolanda Jetten, and Helena R.M. Radke. Cyber-dehumanization: Violent video game play dimishes our humanity. *Journal of Experimental Social Psychology*, 2005. vi, 12

[15] Ecléa Bosi. *O tempo vivo da memória - Ensaios de Psicologia Social*. Atelie Editorial, Paris, 2003. 1

[16] Darryl Charles and Daniel Livingstone. AI: the Missing Link in Digital Game Interface Design? In Matthias Rauterberg, editor, *Entertainment Computing – ICEC 2004 SE - 44*, volume 3166 of *Lecture Notes in Computer Science*, pages 351–354. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22947-6. doi: 10.1007/978-3-540-28643-1_44. URL http://dx.doi.org/10.1007/978-3-540-28643-1_44. 19

[17] Seth Cooper, Adrien Treuille, and Janos Barbero. The challenge of designing scientific discovery games. *... of Digital Games*, pages 40–47, 2010. doi: 10.1145/1822348.1822354. URL http://dl.acm.org/citation.cfm?id=1822354. 18

[18] Dobromir Dochev. Objective evaluation of quality of colour reproduction in current LCD monitors. *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing - CompSysTech '08*, page IIIB.3, 2008. doi: 10.1145/1500879.1500922. 39

[19] Dropbox Inc. Dropbox. URL https://www.dropbox.com/home, acessado em: 13/07/13. 57

[20] Andy Field. *Discovering Statistics using IBM SPSS Statistics*. SAGE, 2013. ISBN 1446274586. 63

[21] Michele J. Fleming and Debra J. Rick Wood. Effects of violent versus nonviolent video games on childrens arousal agressive mood and positive mood. *Journal of Applied Social Psychology*, 10, 2001. vi, 4, 9, 10

[22] Leonardo G De Freitas, Igor Rafael De Sousa, Carla Denise Castanho, Luiggi Monteiro Reffatti, Anderson C Cardoso, and Guilherme N Ramos. Gear2D: An extensible component-based game engine. pages 81–88. 3, 33

[23] Warren Harrop and Grenville Armitage. Modifying first person shooter games to perform real time network monitoring and control tasks. *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06*, page 10, 2006. doi: 10.1145/1230040.1230074. URL http://portal.acm.org/citation.cfm?doid=1230040.1230074. 3, 16

[24] Sylvie Héber, Renée Béland, Odrée Dionne-Fournelle, Martine Crete, and Sonia J. Lupien. Physiological stress response to video-game playing: the contribution of built-in music. *Journal of Applied Social Psychology*, 10, 2012. vi, 11

[25] Jelsoft Enterprises Ltd. The Hive Workshop - Hosted Projects, 2008. URL http://www.hiveworkshop.com/forums/projects.php, acessado em: 01/07/2013. 16

[26] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *2009 IEEE 31st International Conference on Software Engineering*, pages 485–495. IEEE, 2009. ISBN 978-1-4244-3453-4. doi: 10.1109/ICSE.2009.5070547. URL http://dl.acm.org/citation.cfm?id=1555062. 51

[27] Jim Keeley. Protein-folding game taps power of worldwide audience to solve difficult puzzles, 2010. URL http://www.eurekalert.org/pub_releases/2010-08/hhmi-pgt080310.php, acessado em: 2013/04/08. 17

[28] Microsoft. Conditional compilation directives, 2013. URL http://msdn.microsoft.com/en-us/library/aa691099(v=vs.71).aspx, acessado em: 30/06/2013. 50

[29] Mixamo. Soldier Character Pack. URL http://u3d.as/content/mixamo/soldier-character-pack/1sN, acessado em: 13/07/13. vi, 37

[30] Gabe Newell. Gabe Newell writes for the EDGE, 2008. URL http://www.edge-online.com/features/gabe-newell-writes-edge/. 16

[31] NVIDIA Corporation. 30-Bit Color Technology for NVIDIA® Quadro®. 2009. URL http://www.nvidia.com/docs/IO/40049/TB-04701-001_v02_new.pdf. 39

[32] Inc Playfish. SimCity Social. URL https://www.facebook.com/simcitysocial. vi, 17

[33] Stephen E. Siwek. Essential facts about the computer and video game industry. *Entertainment Software Association*, 2010. 1

[34] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, March 2006. ISSN 0885-6125. doi: 10.1007/s10994-006-6205-6. URL http://link.springer.com/10.1007/s10994-006-6205-6. 19

[35] Unity Technologies. Component Class Reference, 2013. URL http://docs.unity3d.com/Documentation/ScriptReference/Component.html, acessado em: 30/06/2013. 53

[36] Unity Technologies. Extending the Editor, 2013. URL http://docs.unity3d.com/Documentation/Components/gui-ExtendingEditor.html, acessado em: 29/06/2013. 48

[37] Unity Technologies. Mecanim, 2013. URL http://unity3d.com/unity/animation/, acessado em: 25/06/2013. 33, 37

[38] Unity Technologies. Asset Store, 2013. URL http://unity3d.com/asset-store/, acessado em: 25/06/2013. 36

[39] Unity Technologies. Using Terrains, 2013. URL http://docs.unity3d.com/Documentation/Components/terrain-UsingTerrains.html, acessado em: 25/06/2013. 36

[40] Unity Technologies. Build Player Pipeline, 2013. URL http://docs.unity3d.com/Documentation/Manual/BuildPlayerPipeline.html, acessado em: 30/06/2013. 69

[41] Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):57, August 2008. ISSN 00010782. doi: 10.1145/1378704.1378719. URL http://portal.acm.org/citation.cfm?doid=1378704.1378719. 17, 18

[42] Jim Whitehead. Fantasy, Farms, and Freemium: What Game Data Mining Teaches Us About Retention, Conversion, and Virality (Keynote Abstract). page 4503, 2011. 16

[43] Sihai Zhang, Zhiwei Song, and Zhi Liang. Behavior statistics and social network analysis of online Go game players. *2011 International Conference on Cloud and Service Computing*, pages 77–82, December 2011. doi: 10.1109/CSC.2011.6138556. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6138556. 16